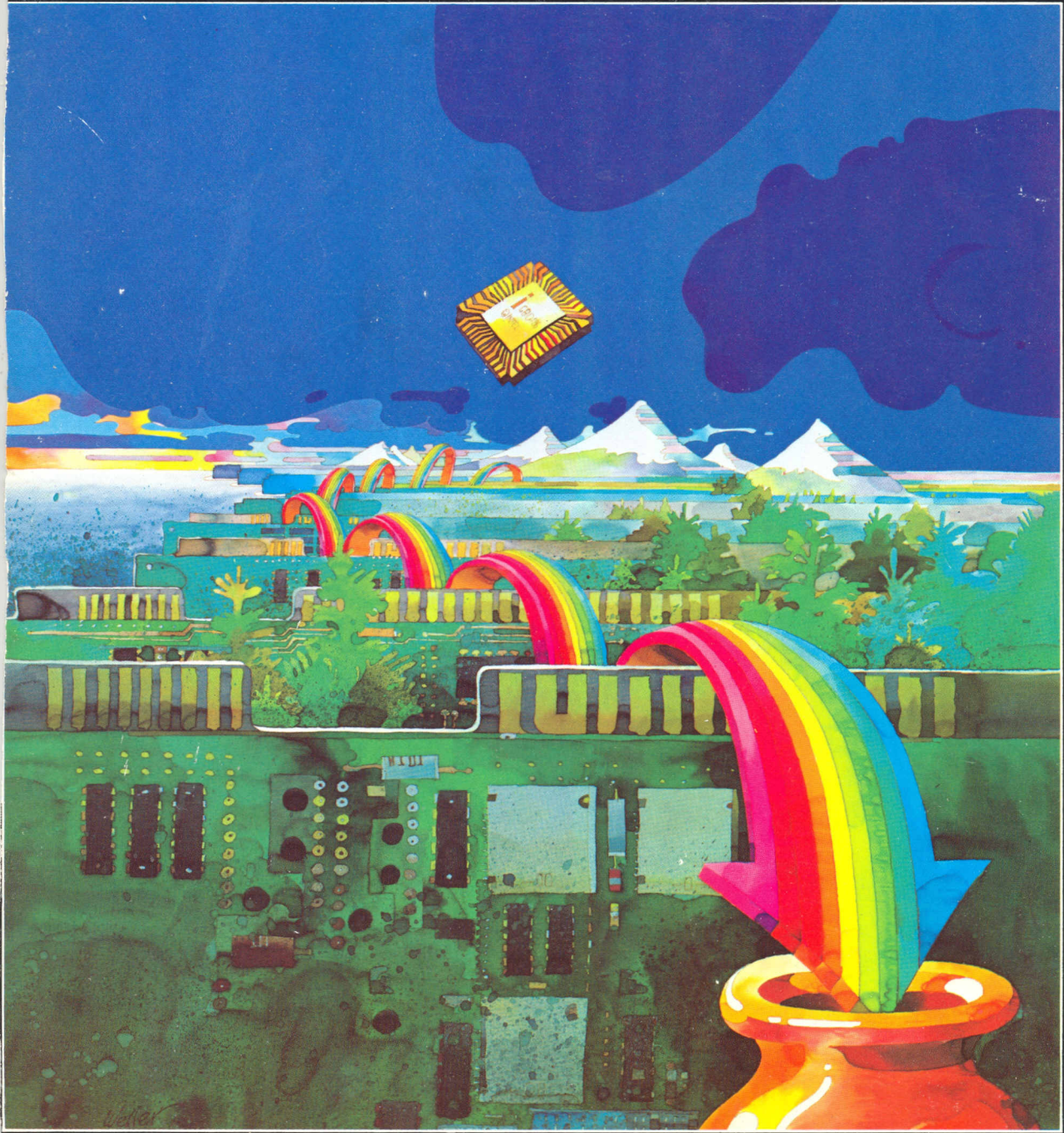


intel

OEM Systems Handbook



Order Number: 210941-001

LITERATURE

1983 will be a year of transition for Intel's catalog program. In order to better serve you, we are reorganizing many of our catalogs to more closely reflect product groups.

In addition to the new product line handbooks listed below, the INTEL PRODUCT GUIDE (Order No. 210846) is available free of charge. This GUIDE serves as a handy reference tool to Intel's complete product line along with information on quality/reliability, packaging and ordering, customer training classes and product services.

Consult the INTEL LITERATURE GUIDE (no charge, Order No. 210620) for a complete listing of Intel literature. Write or call the Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only).

HANDBOOKS

Memory Components Handbook (Order No. 210830)

Contains all application notes, article reprints, data sheets and other design information on RAMs, DRAMs, EPROMs, E²PROMs, Bubble Memories.

Microcontroller Handbook (Order No. 210918)

Contains all application notes, article reprints, data sheets, and other user information on the MCS-48, MCS-51 (8-bit) and the new MCS-96 (16-bit) product families.

Military Handbook (Order No. 210461)

Contains complete data sheets on all military products.

Microprocessor and Peripherals Handbook (Order No. 210844)

Contains data sheets on all microprocessors and peripherals. (Individual User Manuals are also available on the 8085, 8086, 8088, 186, 286, etc.)

Development Systems Handbook (Order No. 210940)

Contains data sheets on development systems and supporting software.

OEM Systems Handbook (Order No. 210941)

Contains all application notes, article reprints and data sheets for OEM boards and systems.



OEM SYSTEMS HANDBOOK

* XENIX is a trademark of Microsoft Corporation.
† Intel is a trademark of Xerox Corporation. 1983

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein. Intel retains the right to make changes to these specifications at any time, without notice. Contact your local sales office to obtain the latest specifications before placing your order. Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASP 7-104.9(a)(2). Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

i8086	IntelLink	i8080
i8088	i8087	i8085
MultiChannel	i8086	i8085
MULTIUS	i8086	i8085
MULTIMODULE	i8086	i8085

and the combinations of ICE, i8086, MCS or RMX and a numerical suffix. Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051



OE M SYSTEMS HANDBOOK

* XENIX is a trademark of Microsoft Corporation.

** Ethernet is a trademark of Xerox Corporation.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9). Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

iCS	Intellink	iSDM
iLBX	iOSP	iSXM
iMMX	iRMX	Multichannel
Insite	iSBC	MULTIBUS
Intel	iSBX	MULTIMODULE

and the combinations of ICE, iCS, iSBC, MCS or RMX and a numerical suffix.

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

CHAPTER 1

System Packaging and Power Supplies and Service

DATA SHEETS

iSBC 680/681 Multistore User System Package	1-1
iSBC 665/645 System Chassis and Power Supply	1-5
iSBC 661 System Chassis	1-9
iSBC 660 System Chassis	1-13
iSBC 640 Power Supply	1-16
iSBC 635 Power Supply	1-18
iSBC 608/618 Cardcages	1-21
iSBC 604/614 Modular Cardcage/Backplane	1-25
iMBX 100/110/120 MULTIBUS Exchange Hardware Subscription Service	1-27

CHAPTER 2

Integrated Microcomputer Systems

DATA SHEETS

iSXM 100 System 86/300 Series XENIX* System Extension Module	2-1
System 86/380X Microcomputer System XENIX*	2-4
System 86/380 Microcomputer System iRMX 86 Operating System	2-12
System 86/330X Microcomputer System XENIX*	2-20
System 86/330A Microcomputer System iRMX 86 Operating System	2-27

FACT SHEETS

iTPS 86/445 Transaction Processing System	2-34
iTPS 86/435 Transaction Processing System	2-39

CHAPTER 3

Single Board Computers

DATA SHEETS

MULTIBUS System Bus	3-1
MULTICHANNEL I/O Bus	3-11
iLBX Execution Bus	3-21
iSBX I/O Expansion Bus	3-29
iSBC 286/10 Single Board Computer	3-39
iSBC 88/45 Advanced Data Communications Processor Board	3-48
iSBC 88/40 Measurement and Control Computer	3-57
iSBC 88/25 Single Board Computer	3-65
iSBC 86/30 Single Board Computer	3-73
iSBC 86/14 Single Board Computer	3-73
iSBC 86/12A (or pSBC 86/12A) Single Board Computer	3-81
iSBC 86/05 Single Board Computer	3-89
iSBC 80/30 (or pSBC 80/30) Single Board Computer	3-97
iSBC 80/24 (or pSBC 80/24) Single Board Computer	3-105
iSBC 80/20-4 (or pSBC 80/20-4) Single Board Computer	3-114
iSBC 80/16 Single Board Computer	3-121
iSBC 80/10B (or pSBC 80/10B) Single Board Computer	3-128
iSBC 80/05 (or pSBC 80/05) Single Board Computer	3-135

CHAPTER 3 (continued)

Single Board Computers

APPLICATION NOTES

- AP-28A Intel MULTIBUS Interfacing 3-140
- AP-114 Using the iSBC 88/40 Measurement and Control Computer in PID Applications... 3-180

ARTICLE REPRINTS

- AR-48 Reduce Your uC-Based System Design Time by Using Single Board Computers 3-202
- AR-55 Design Motivations for Multiple Processor Microcomputer Systems 3-214
- AR-65 Triple-Bus Architecture on a Single-Board Microcomputer 3-224
- AR-72 16-Bit Single Board Computer Maintains 8-Bit Family Ties 3-231
- AR-122 A New Family of MULTIMODULE Boards Extends the Solutions Provided
by Intel's Single Board Computer 3-238
- AR-123 Special Function Modules Ride on Computer Board 3-242
- AR-133 Multiprocessing System Mixes 8- and 16-Bit Microcomputers 3-249
- AR-229 Enhanced uComputer Boards Strengthen Factory and Office Controllers 3-258

CHAPTER 4

High Speed Math Boards

DATA SHEETS

- iSBC 337 MULTIMODULE Numeric Data Processor 4-1
- iSBX 332 Floating Point Math MULTIMODULE Board 4-9
- iSBX 331 Fixed/Floating Point Math MULTIMODULE Board 4-14

CHAPTER 5

Systems Software

DATA SHEETS

- iSBC 957B iAPX 86, 88 Interface and Execution Package 5-1
- iMMX 800 MULTIBUS Message Exchange Software 5-6
- iRMX 286R Operating System 5-10
- iSDM 286 System Debug Monitor 5-14
- iRMX 88 Real-Time Multi-Tasking Executive 5-17
- iRMX 86 Operating System 5-23
- Preconfigured iRMX 86 Operating System 5-43
- iRMX 86 Languages and Utilities 5-49
- iOSP 86 iAPX 86/30 and iAPX 88/30 Support Package 5-53

APPLICATION NOTES

- AP-43 Using the iSBC 957 Execution Vehicle for Executing 8086 Program Code 5-57
- AP-86 Using the iRMX 86 Operating System 5-87
- AP-109 Using Intel Single Board Computers for Serial Distributed Processing Links 5-143
- AP-110 Using the iRMX 86 Operating System on iAPX 86 Component Designs 5-211

ARTICLE REPRINTS

- AR-124 Introducing the RMX/86 Real-Time, Multitasking, 16-Bit Operating System 5-269
- AR-125 Modular Multitasking Executive Cuts Cost of 16-Bit OS Design 5-273
- AR-172 Multitasking Executive Speeds 16-Bit Micros 5-279
- AR-194 Choosing a VLSI-Compatible System 5-285
- AR-195 Microcomputer Operating System Trends 5-288
- AR-196 The iRMX 86 Operating System 5-294
- AR-220 An Object Oriented Operating System for Microcomputers 5-302
- AR-236 Let Operating Systems Aid in Component Design 5-309

CHAPTER 6

Memory Expansion Boards

A Primer on Magnetic Bubble Memory	6-1
DATA SHEETS	
iSBC 464 64K-Byte EPROM Expansion Board	6-15
iSBC 428 Universal Site Memory Expansion Board	6-18
iSBC 341 28-Pin EPROM MULTIMODULE	6-22
iSBC 340 (or pSBC 340) 16K Byte EPROM MULTIMODULE	6-24
iSBC 304 128K-Byte RAM MULTIMODULE Board	6-28
iSBC 303 Parity MULTIMODULE	6-31
iSBC 302 8K-Byte RAM MULTIMODULE	6-35
iSBC 301 4K-Byte RAM MULTIMODULE	6-37
iSBC 300A 32K-Byte RAM MULTIMODULE	6-28
iSBC 300 (or pSBC 300) 32K-Byte RAM MULTIMODULE	6-24
iSBC 254S Bubble Memory Board	6-40
iSBX 251 and 251C Bubble Memory MULTIMODULE Board	6-43
iSBC 064 RAM Memory Board	6-48
iSBC 032A/064A/028A/056A RAM Parity Memory Boards	6-50
iSBC 028CX/056CX/012CX RAM ECC Memory Boards	6-53
iSBC 028C/056C/012C RAM ECC Memory Boards	6-58
iSBC 012B RAM Parity Memory Board	6-62
ARTICLE REPRINTS	
AR-202 Choose the Right Level of Memory-Error Protection	6-64

CHAPTER 7

Peripheral Controllers

DATA SHEETS	
iSBX 275 Video Graphics Controller	7-1
iSBX 270 Video Display Controller	7-5
iSBC 220 SMD Disk Controller	7-9
iSBX 218 Flexible Disk Controller	7-13
iSBC 217B Cartridge Tape Drive Interface	7-17
iSBC 215 Generic Winchester Controller	7-20
iSBC 208 Flexible Disk Controller	7-25
iSBC 204 Single Density Flexible Diskette Controller	7-29
APPLICATION NOTES	
AP-118 Raster Graphics Techniques for MULTIBUS Users	7-32

CHAPTER 8

Communications Controllers

DATA SHEETS	
iNA 950-1 Local Area Network Software	8-1
IDCM 911-1 Intellink Ethernet** Cluster Module	8-6
iSBC 589 Intelligent DMA Controller	8-8
iSBC 580 MULTICHANNEL to iLBX Bus Interface	8-15
iSBC 570, 576, 577 Intel Speech Transaction Family	8-19
iSBC 550 Ethernet** Communications Controller	8-34
iSBC 544 Intelligent Communications Controller	8-38
iSBC 534 Four Channel Communications Expansion Board	8-45
iSBC 88/45 Advanced Data Communications Processor Board	8-49
APPLICATION NOTES	
AP-53 Using the iSBC 544 Intelligent Communications Controller	8-58
AP-149 Using the iSBC 550 Ethernet** Communications Controller	8-120
ARTICLE REPRINTS	
AR-69 Dual-Port RAM Hikes Throughput in Input/Output Controller Board	8-145

CHAPTER 9

Digital I/O Expansion and Signal Conditioning Boards

1-8	DATA SHEETS	
	iSBC 569 Intelligent Digital Controller	9-1
8-18	iSBC 556 Optically Isolated I/O Board	9-6
8-18	iSBC 519 (or pSBC 519) Programmable I/O Expansion Board	9-8
8-22	iSBC 517 Combination I/O Expansion Board	9-12
8-24	iSBX 488 GPIB MULTIMODULE Board	9-16
8-28	iSBX 352 Bit Serial Communications MULTIMODULE Board	9-20
8-31	iSBX 351 Serial I/O MULTIMODULE Board	9-26
8-35	iSBX 350 Parallel I/O MULTIMODULE Board	9-32
8-37	APPLICATION NOTES	
8-38	AP-96 Designing iSBC MULTIMODULE Boards	9-36

CHAPTER 10

ICS Industrial Control Series and Analog I/O Expansion

8-43	DATA SHEETS	
8-48	ICS 910/920/930 Signal Conditioning/Termination Panels	10-1
8-50	iSBX 328 Analog Output MULTIMODULE Board	10-9
8-52	iSBX 311 Analog Input MULTIMODULE Board	10-13
8-55	ICS 80 Industrial Chassis	10-17
8-58	APPLICATION NOTES	
8-60	AP-52 Using Intel's Industrial Control Series in Control Applications	10-22
	AP-60 Closed Loop Control Using the iSBC 569/941 Intelligent Digital	10-77
	ARTICLE REPRINTS	
	AR-91 Designing and Assembling Microcomputer Systems Grows Easier	10-137

CHAPTER 11

7-8	Insite User's Program Library	11-1
-----	-------------------------------	------

CHAPTER 12

Development Systems Operation

7-10	DATA SHEETS	
7-17	Digital Research Inc. CP/M 2.2. Operating System	12-1
7-22	Digital Research Inc. CP/M-86 Operating System	12-4
7-23	Microsoft MACRO-80 Utility Software Package	12-7
7-25	Microsoft BASIC 80 Interpreter Software Package	12-9
7-28	Microsoft COBOL 80 Software Package	12-12
	Microsoft FORTRAN 80 Software Package	12-17
	Microsoft BASIC 86 Interpreter Software Package	12-20
	Jovial 86 Cross-Compiler	12-23

Power Supplies and Service



System Packaging and Power Supplies and Service

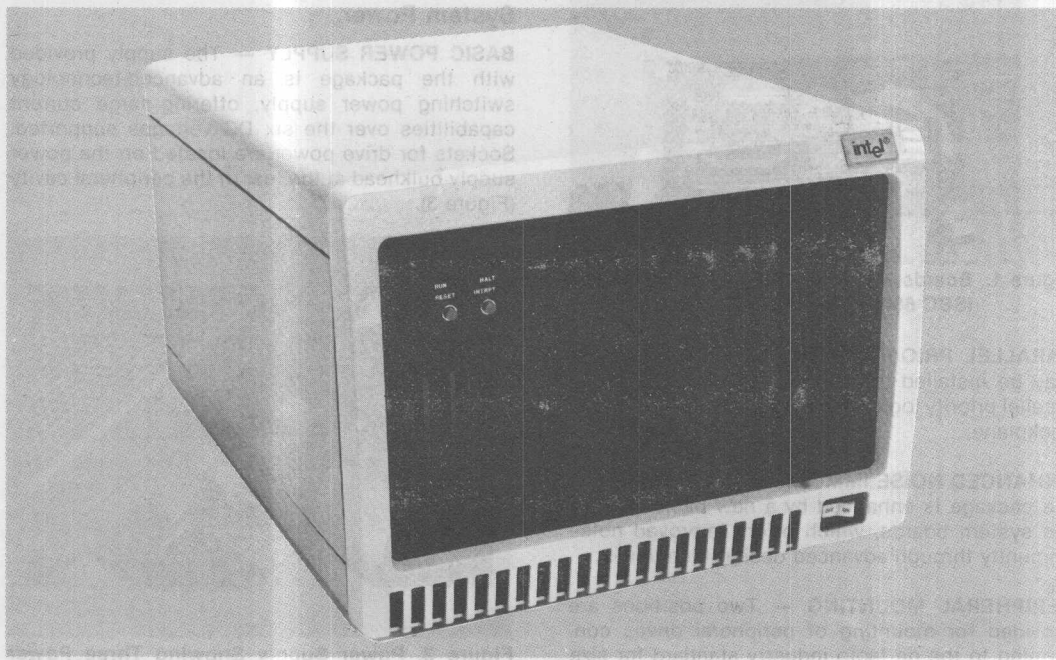
1



iSBC™ 680/iSBC 681 MULTISTORE™ USER SYSTEM PACKAGE

- Complete system package for user-selected Intel® iSBC boards and up to two 8" peripheral drives
- Holds up to six iSBC boards compatible with the MULTIBUS® system bus
- Supplies ± 5 , ± 12 , ± 24 VDC to power boards and peripheral drives
- Available in table-top (iSBC 680 package) or rack-mount (iSBC 681 package) configurations
- Designed to meet UL safety requirements and FCC/VDE EMI limits
- Power supply provides 8 ms of power-fail warning, plus a real-time clock

The iSBC 680/iSBC 681 Multistore User System Package products make available to the OEM a new way to assemble his systems for those applications requiring rotating memory or other peripherals built in the 8" industry-standard form factor. The Multistore package allows the OEM to select the iSBC boards required for the job, and to independently choose from a wide variety of peripherals to complete the system. The switching power supply provides sufficient current at all voltage levels to power most manufacturers' drives, as well as furnishing the standard MULTIBUS system bus voltages to the iSBC boards in the package's cardcage. The appearance of the packages provides an attractive addition to the OEM's system, while the construction allows easy access to the interior for service.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, MULTIBUS, iRMX, iSBC, iSBX, MULTIMODULE, Multistore and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

FUNCTIONAL DESCRIPTION

Physical Packaging

PACKAGE CONSTRUCTION — The Multistore package is constructed entirely of metal, and all cover pieces are gasketed to completely contain high-frequency noise from the power supply and the system boards within the package.

MOUNTING OPTIONS — The iSBC 680 package is a table-top structure; the iSBC 681 package is the rack-mount version with slides attached to the side panels and a wider trim bezel to cover the mounting rails.

COOLING — The boards and peripherals installed in the package are cooled by air brought in at the bottom of the front panel and drawn through the power supply, with the heated air discharged to the rear of the package.

CARDCAGE/BACKPLANE — The cardcage/backplane accepts up to six iSBC boards compatible with the Intel MULTIBUS system bus (Figure 1).

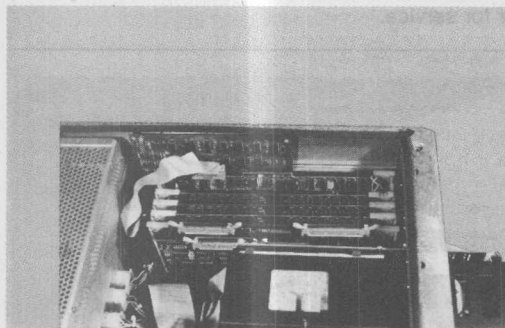


Figure 1. Boards Mounted in the Cardcage of the iSBC 680 Package

PARALLEL PRIORITY — Up to six bus masters may be installed in the package because of the parallel priority logic which is an integral part of the backplane.

ENHANCED NOISE IMMUNITY — The integrity of the package is enhanced by a new backplane for the system boards, which offers improved noise immunity through advanced design techniques.

PERIPHERAL MOUNTING — Two positions are provided for mounting of peripheral drives conforming to the de facto industry standard for size and mounting points on 8" peripherals. The mounting system provides slides for the bases of the

drives, allowing the drives to be installed/removed from the front of the package (Figure 2).

Blank covers with ventilation slots are provided with the package for the unused peripheral positions and for those peripherals not furnished with a cosmetic cover.

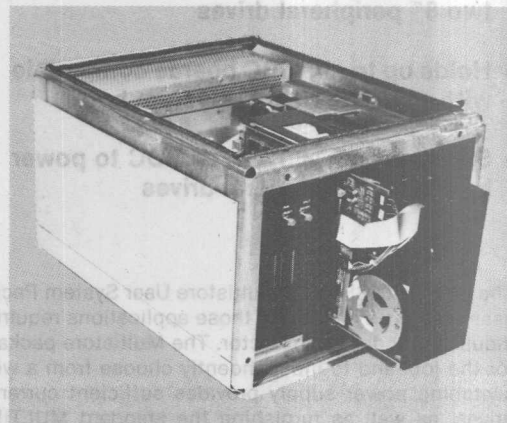


Figure 2. iSBC 680 Package with Winchester Drive Pulled Out for Servicing

System Power

BASIC POWER SUPPLY — The supply provided with the package is an advanced-technology switching power supply, offering large current capabilities over the six DC voltages supported. Sockets for drive power are located on the power supply bulkhead at the rear of the peripheral cavity (Figure 3).

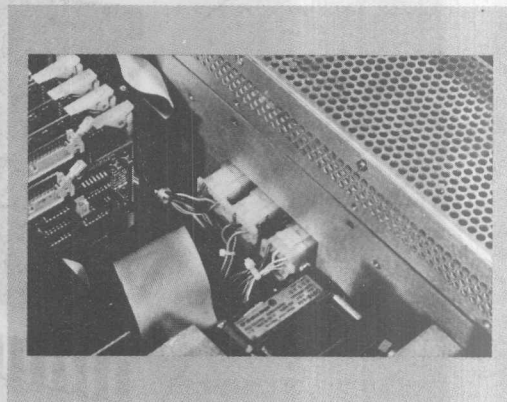


Figure 3. Power Supply Showing Three Power Connectors (Two for Peripherals, One for the Cardcage/Backplane)

INTERNATIONAL ACCEPTANCE — The package is a UL-recognized component, and it has been designed to meet the safety requirements of CSA and VDE.

MEETS EMI STANDARDS — The FCC standards for conducted and radiated EMI (electromagnetic interference), as well as the VDE requirements (0871/0875), are met by the package.

POWER-FAIL/AUTO-RESTART SYSTEM — The package gives the user a set of logic signals providing advanced warning of power failures and protection for battery backed-up memory as the DC voltages fall. It also furnishes a real-time clock derived from the line frequency, thus ensuring long-term stability in user time-keeping.

User Interface

USER CONTROLS — At the front of the package the user has access to both the AC power switch (with integral circuit breaker) and controls and indicators for the microcomputer system itself. "RESET" and "INTERRUPT" switches are provided, along with "RUN" and "HALT" LED indicators.

DEVICE INTERFACE — With the package designed for maximum suppression of both EMI and ESD (electrostatic discharge) the preferred interface between the installed boards and external devices is with shielded cables isolated through user-supplied connectors installed in the panel provided at the rear of the package (Figure 4). The six cut-outs, sized for 50-pin connectors, are furnished with individual cover plates.

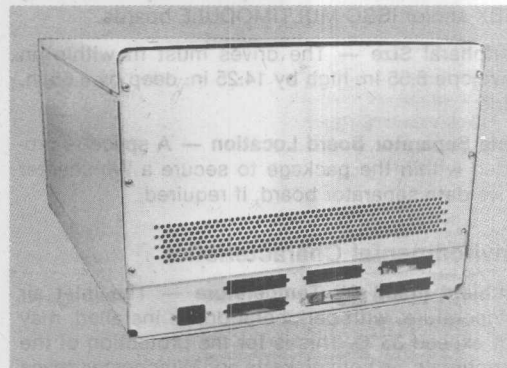


Figure 4. iSBC 680 Package Showing External Device Connection Panel

SPECIFICATIONS

Input Power

Frequency — 47-66 Hz

Voltage — 90-126 VAC/180-252 VAC, single phase (user-selectable).

Periodic and Random Deviation (PARD) — 50 mV peak-to-peak, all outputs.

Output Regulation (Combined Line and Load) — $\pm 1\%$ under any conditions of AC input voltage variation (within operational range) and output load change.

Line Transient Tolerance — A signal of up to 1000 VDC, with a pulse width of up to 50 μ s, will have no effect on system operation.

Output Power

Power Fail Indication — PFIN/ is generated approximately 8 ms after the input drops below 90/180 VAC. PFIN/ is available on the P2 connector of the cardcage/backplane for generation of an interrupt. The ± 24 VDC outputs will go to zero within

1 ms of issuance of PFIN/; the ± 5 and ± 12 VDC outputs will remain within specification for at least 8 ms, after which MPRO/ will go true to protect non-volatile memories from being written into as DC power fails.

System Clock — The power supply provides a 2x line frequency clock output, available on the P2 connector of the cardcage/backplane.

Nominal Voltage	Current ¹ (Max Amps)	Typical Peripheral Power Requirements ²	
		8" Winchester	Diskette Drive
+ 5	30.0	5.1	1.0
- 5	2.0	0.25	0.07
+ 12	2.9	—	—
- 12	3.0	0.7	—
+ 24	7.8	4.0	1.8
- 24	1.6	—	—

NOTES:

1. The maximum power available from the supply, from all outputs, is 300 watts.
2. These are worst-case data, drawn from manufacturers' data sheets.

Physical Characteristics (Figure 5)

Width — 16.8/19.0 in. (42.5/48.3 cm)

iSBC 680/iSBC 681 Packages

Length — 21.5 in. (54.6 cm)

Height — 12.2 in. (31.1 cm)

Weight — 40 lb (18.2 kg) (approximate)

Board Slots — Six @ 0.665 in. on centers between boards. The board in the top slot may contain any iSBX and/or iSBC MULTIMODULE boards.

Peripheral Size — The drives must fit within an envelope 8.55 in. high by 14.25 in. deep by 4.65 in. wide.

Data Separator Board Location — A space is provided within the package to secure a Winchester drive data separator board, if required.

Environmental Characteristics

Ambient (Inlet) Air Temperature — The inlet air temperature, with peripheral drives installed, may not exceed 35°C. This is for the protection of the peripherals, as both diskette and Winchester drives have ambient maximums of 40°C in most instances.

Humidity — 20% to 80% RH, non-condensing for the chassis and typical peripheral content.

NOTE: The photos of the Multistore packages in this data sheet show boards, an 8" Winchester hard disk drive, and an 8" flexible disk drive installed. The packages *do not* include these boards and peripherals; they are shown in the photographs to illustrate physical arrangements in the Multistore package.

Equipment Supplied

iSBC 680 Chassis — Includes table-top package with aluminum sheet metal, 6-slot cardcage/backplane, combination On/Off switch and circuit breaker, peripheral mounting hardware (user must supply power and signal cabling for peripherals), and power supply with AC power cord.

iSBC 681 Chassis — Same as iSBC 680 chassis, plus rack-mount chassis slides and wider bezel.

Reference Manual

162432 — iSBC™ 680/681 Multistore™ Chassis Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

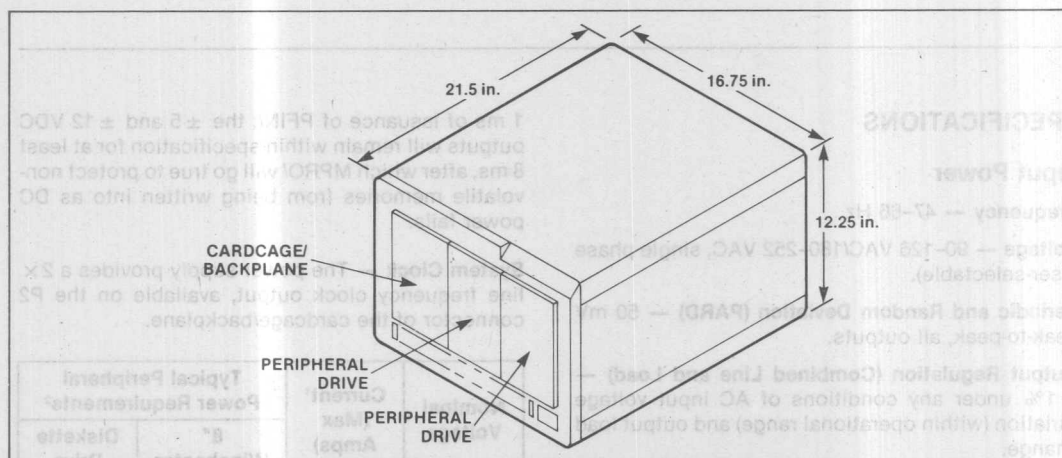


Figure 5. Physical Dimensions of the iSBC 680 Multistore User System Package

ORDERING INFORMATION

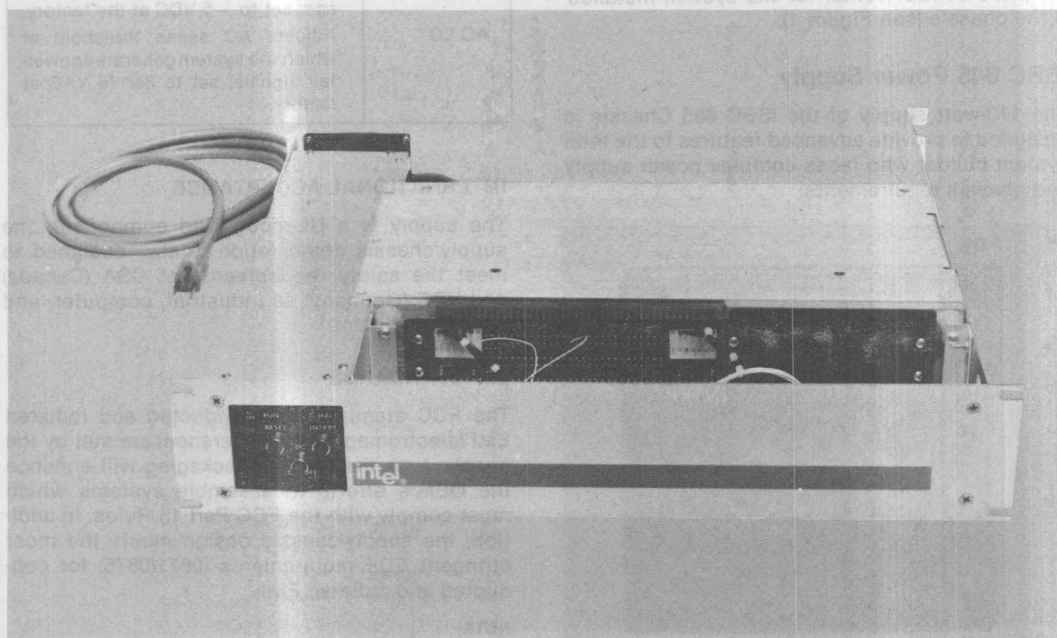
Part Number	Description
SBC 680	Multistore User System Package (Table-Top)
SBC 681	Multistore User System Package (Rack-Mount)



iSBC 665/iSBC 645™ SYSTEM CHASSIS AND POWER SUPPLY

- Intel MULTIBUS™ system bus 4-slot packaging
- Complete package of rack-mounting, cooling, controls, and power
- Advanced 110 watt switching power supply generates ± 5 , ± 12 VDC
- Meets U.S. and International EMI and safety requirements
- Wide AC voltage margins keep systems running during "brownouts"
- Front panel switches, indicators, and adjustments for operational and service convenience
- Power sense circuitry interrupts system 6 msec prior to power failure

The Intel iSBC 665/iSBC 645 Chassis system provides the MULTIBUS system bus user with a compact set of products offering new standards in 4-slot rack-mount packaging. A high-efficiency switching power supply allows use of 115/230 VAC (+15/-20%), with large surge and noise components, to deliver smooth, stable DC power to the OEM board load. Advanced power-fail sense and restart logic gives the user sufficient time to bring the system to an orderly shutdown in the event of AC mains power failure. Mechanical design features include EMI suppression and a retainer/cover for system boards and I/O edge connectors.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: CREDIT, Index, Intel, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, μ Scope, Promware, MCS, ICE, IRMX, iSBC, ISBX, MULTIMODULE and ICS, and the combination of MCS, ICE, iSBC, ISBX, IRMX or ICS, and a numerical suffix. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

FUNCTIONAL DESCRIPTION

iSBC 665™ System Chassis

The iSBC 665 Chassis is a complete micro-computer package providing four board slots in a 3.5" vertical space.

RACK MOUNT PACKAGE

The iSBC 665 Chassis mounts in a 19" EIA standard rack, using its front panel and hangers at the rear of the chassis to secure it to both sets of rails in the cabinet. If slide mounting is preferred, a tray with slides should be used as a platform for the chassis. The physical integrity of the system is enhanced by addition of a connector retainer at the (rear-facing) opening of the cardcage.

INTEGRAL COOLING

The fan on the power supply is utilized to draw ambient air across the boards prior to its being used to cool the supply.

FRONT PANEL

The front panel of the iSBC 665 Chassis forms a complete control center for the system installed in the chassis (see Figure 1).

iSBC 645 Power Supply

The 110-watt supply of the iSBC 665 Chassis is designed to provide advanced features to the Intel system builder who faces complex power supply and chassis requirements.

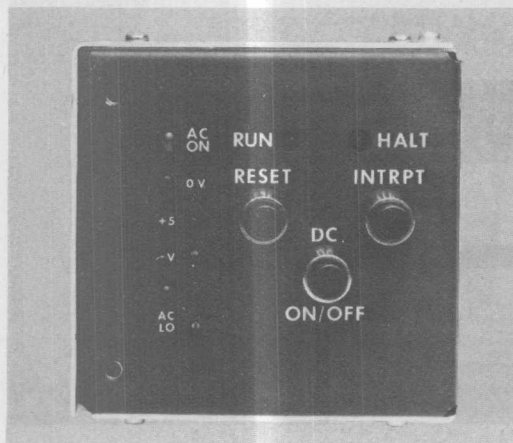


Figure 1. iSBC 665™ Chassis Front Panel Controls

Table 1. iSBC 665™ Chassis/iSBC 645™ Power Supply Control Panel Functions

Label	Function
Controls	
DC ON/OFF	Controls all DC power to the chassis.
RESET	Generates RESET/ signal to pin 14 of P1 (MULTIBUS system bus) backplane.
INTRPT	Generates INT/ signal to pin 42 of P1.
Indicators	
HALT, RUN	Indicate status of system CPU board.
AC ON	Indicates AC power present in supply (AC power switch is located at the rear of the supply).
OV	Indicates power supply shut-down due to an overvoltage condition on +5 or ± 12 VDC outputs.
AC LO	Indicates that AC voltage is below the operating range and the supply has shut down.
Adjustments	
+5	Adjusts +5 VDC output voltage.
-V	Adjusts negative adjustable voltage; set to -5 VDC at the factory.
AC LO	Adjusts AC sense threshold at which the system generates power-fail signals; set to 88/176 VAC at factory.

INTERNATIONAL ACCEPTANCE

The supply is a UL-recognized component; the supply/chassis combination is also designed to meet the safety requirements of CSA (Canada) and VDE (Germany) as industrial, computer, and office equipment.¹

EMI STANDARDS

The FCC standards for conducted and radiated EMI (electromagnetic interference) are met by the supply, thus the chassis packaging will enhance the OEM's efforts to assemble systems which must comply with the FCC Part 15 Rules. In addition, the supply/chassis design meets the most stringent VDE requirements (0871/0875) for conducted and radiated EMI.¹

NOTE:

1. CSA and VDE testing have not been officially completed; testing at independent laboratories indicates that these safety and EMI requirements are met. Official testing should be completed in late 1981.

BROWNOUT PROTECTION

The wide AC voltage input range allows micro-computer systems packaged in the chassis to function normally at extremely low AC voltage supply levels.

POWER-FAIL WARNING AND RECOVERY

In the event of a complete power failure, an interrupt is generated 6 ms prior to the supply's issuing a subsequent memory protect signal, giving sufficient time for execution of a user program to bring the entire system to an orderly shut-down.

POWER TRANSIENT TOLERANCE

The supply provides immunity for the system from the high-voltage transient surges and spikes seen in AC power systems. The supply itself provides this isolation with a metal-oxide varistor (MOV) and line filter in the input circuitry.

POWER LINE CLOCK

A clock signal is developed from the AC line at twice the line frequency; this gives the system user an extremely accurate time base.

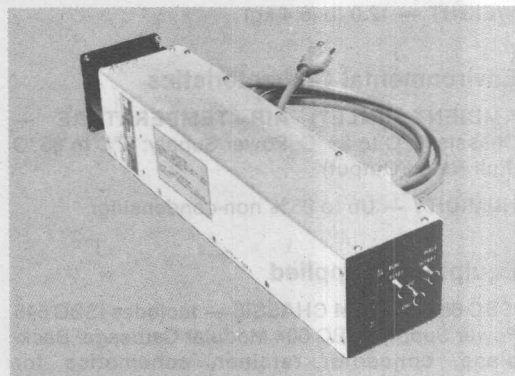


Figure 2. ISBC 645™ Power Supply

SPECIFICATIONS**Electrical Characteristics****INPUT POWER**

Frequency: 47-66 Hz

Voltage: 115/230 VAC Single Phase

Range: 90 to 126 VAC/180 to 252 VAC

Consumption (Max.): 230 watts

OUTPUT POWER

Nominal Voltage	Current ² (Max. Amps)	Current Limit Point (Amps)	Overvoltage Protection ³
+ 5	15	18.75	5.25 to 6.25
+ 12	3	3.75	12.6 to 15.0
- 12	1	1.25	- 12.6 to - 15.0
- Adjustable ⁴	1	1.25	N/A

NOTES:

- Total output power is 110 watts; a maximum of 128 watts is available, but proper operation of the power-fail circuitry is not guaranteed above 110 watts.
- A minimum load is required on the + 5 VDC output; this load must be at least 1/2 the sum of the loads (in watts) of the remaining three outputs.
- 2.5 to - 12 VDC; factory set to - 5 VDC.

OUTPUT REGULATION (COMBINED LINE AND LOAD) — $\pm 1\%$ under any conditions of AC mains voltage variation (within operational range) and output load change.

PERIODIC AND RANDOM DEVIATION (PARD) — 50 millivolts peak-to-peak, all outputs.

LINE TRANSIENT TOLERANCE — A signal of up to 1000 VDC, with a pulse width of up to 50 microseconds, will have no affect on operation.

POWER FAIL INDICATION — An AC low condition generates ACLO and PFIN/ after AC voltage drops below the allowed voltage range. These signals are available on the P2 connector to generate interrupts. The DC voltages will remain within specifications for 6 milliseconds (worst case) following these interrupts, after which Memory Protect (MPRO/) will go true.

OUTPUT VOLTAGE TEMPERATURE COEFFICIENT — 0.03% per °C over the operating range.

SYSTEM CLOCK — $2 \times$ line frequency clock signal available on P2 connector.

Physical Characteristics (See Figure 3)

WIDTH — 19.0 in. (48.3 cm)

LENGTH — 16.25 in. (41.3 cm)

HEIGHT — 3.5 in. (8.9 cm)

WEIGHT — 12.0 lb (5.4 kg)

Environmental Characteristics
AMBIENT (INLET) AIR TEMPERATURE —
Chassis: 0°C to 55°C; Power Supply: 0°C to 65°C
(Full Rated Output)

HUMIDITY — Up to 95% non-condensing.

Equipment Supplied
iSBC 665 SYSTEM CHASSIS — Includes iSBC 645 Power Supply, iSBC 604 Modular Cardcage/ Backplane, connector retainer, schematics for cardcage/backplane, chassis, and power supply.

iSBC 645 POWER SUPPLY — Includes power supply, schematics for supply.

Reference Manuals (Not Supplied)

142836 — iSBC 665™ System Chassis Hardware Reference Manual

ORDERING INFORMATION

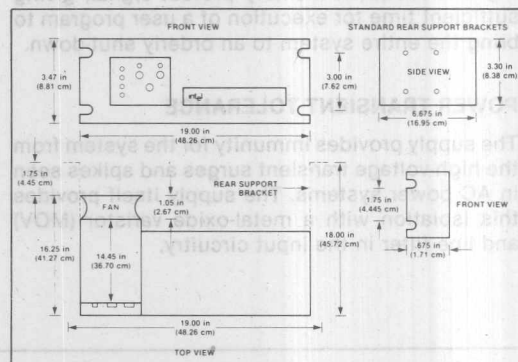
Part Number	Description
SBC 665	System Chassis
SBC 645	Power Supply (110 Watt)

142918 — iSBC 645™ Power Supply Hardware Reference Manual

9800708 — iSBC 604/614™ Modular Cardcage/ Backplane Hardware Reference Manual

9088683 — Intel MULTIBUS Specification

Manuals may be ordered from any Intel sales representative, distribution office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.


Figure 3. iSBC 665™ System Chassis Physical Dimensions

Output Voltage	Current Limit (Max. Amps)	Current Limit Point (Amps)	Overvoltage Protection
+5	18.75	18.75	5.25 to 7.25
+12	3.75	3.75	12.5 to 15.0
-12	1.25	1.25	12.5 to 15.0
Not Available	1.25	1.25	N/A

NOTES:

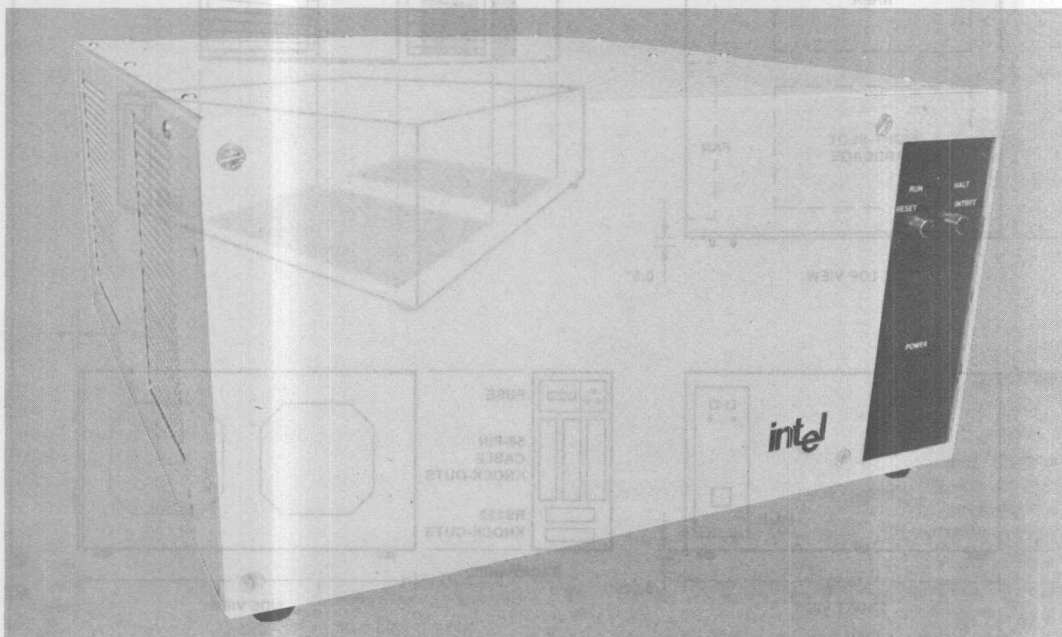
1. Total output power is 110 watts; a maximum of 125 watts is available, but proper operation of the power-fail circuitry is not guaranteed above 110 watts.
2. A minimum load is required on the +5 VDC output; this load must be at least the sum of the loads (in watts) of the remaining three outputs.
3. -12 VDC factory set to -2 VDC.

iSBC® 661 SYSTEM CHASSIS

- Eight-slot MULTIBUS® chassis with parallel priority circuitry
- UL, FCC and CSA approved for data processing equipment
- 230 watt power supply with power fail warning
- Designed for slide rack mounting or bench-top use
- Extra-wide cardcage slot spacing for iSBX™ MULTIMODULE™ board clearance
- Configurable for front or rear access to MULTIBUS® circuit boards
- Five connector ports for I/O cabling
- Operational from 47 Hz to 63 Hz, 100/120/220/240 VAC $\pm 10\%$

The iSBC® 661 System Chassis is one of the most advanced MULTIBUS® (IEEE 796) packaging products. It incorporates unique usability and service features not found on competitive products. This chassis is designed for rack-mount or bench-top applications and reliably operates between the ambient temperatures of 0°C to 50°C. Additionally, this system chassis is certified by UL, CSA, and FCC for data processing equipment.

An application requiring multiprocessing will find this eight-slot MULTIBUS chassis particularly well suited to its needs. Parallel priority bus arbitration circuitry has been integrated into the backplane. This permits a bus master to reside in each slot. Extra-wide inter-slot spacing on the cardcage allows the use of plug-on MULTIMODULE™ boards without blocking adjacent slots. For this reason, the iSBC 661 System Chassis provides the slot-functionality of most 16-slot chassis. Standard logic recognizes a system AC power failure and generates a TTL signal for use in power-down control. Additionally, current limiting and over-voltage protection are provided at all outputs.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, IMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1983

FUNCTIONAL DESCRIPTION

Mechanical Features

The iSBC 661 System Chassis houses, cools, powers, and interconnects up to eight iSBC single board computers and their MULTIMODULE boards for the MULTIBUS System Bus. Based on Intel's iSBC 608 Cardcage, the chassis provides 0.8 inches of board center-to-center clearance on six slots, and 1.2 inches or more of center-to-center clearance on two slots. This permits the users of standard MULTIMODULE boards and custom wire-wrap boards to plug into the MULTIBUS System Bus without blocking adjacent slots. All slots provide enough clearance for iSBC MULTIMODULE boards, and two slots can accommodate iSBX MULTIMODULE boards.

High-technology MULTIBUS applications requiring rack-mount, or laboratory bench-top use will find the iSBC 661 System Chassis ideal. Standard 19" slide-rack mounting is possible with user-provided slides attached to the side panels. Slide mounting holes are provided on the chassis for the slide-rails listed under

User Supplied Options. Rubber feet are included on the chassis for convenient bench-top use.

The chassis is constructed of burnished aluminum which has been coated with corrosion-resistant chromate. It contains a system control module which presents the front panel control switches to the user, and holds the I/O cabling bulkhead to the rear. The chassis has the unique feature of being configurable for either front or rear access to MULTIBUS circuit boards.

This is accomplished by a simple procedure involving removal of the system control module, reversing it end-for-end, and re-securing it to the chassis. The system chassis is shipped in a configuration such that the MULTIBUS boards are installed from the front.

Electrical Features

The iSBC 661 System Chassis is powered by the iSBC 640 power supply. This is a standard Intel power supply which has been adopted by several MULTIBUS vendors throughout the industry. It supplies 230 watts of

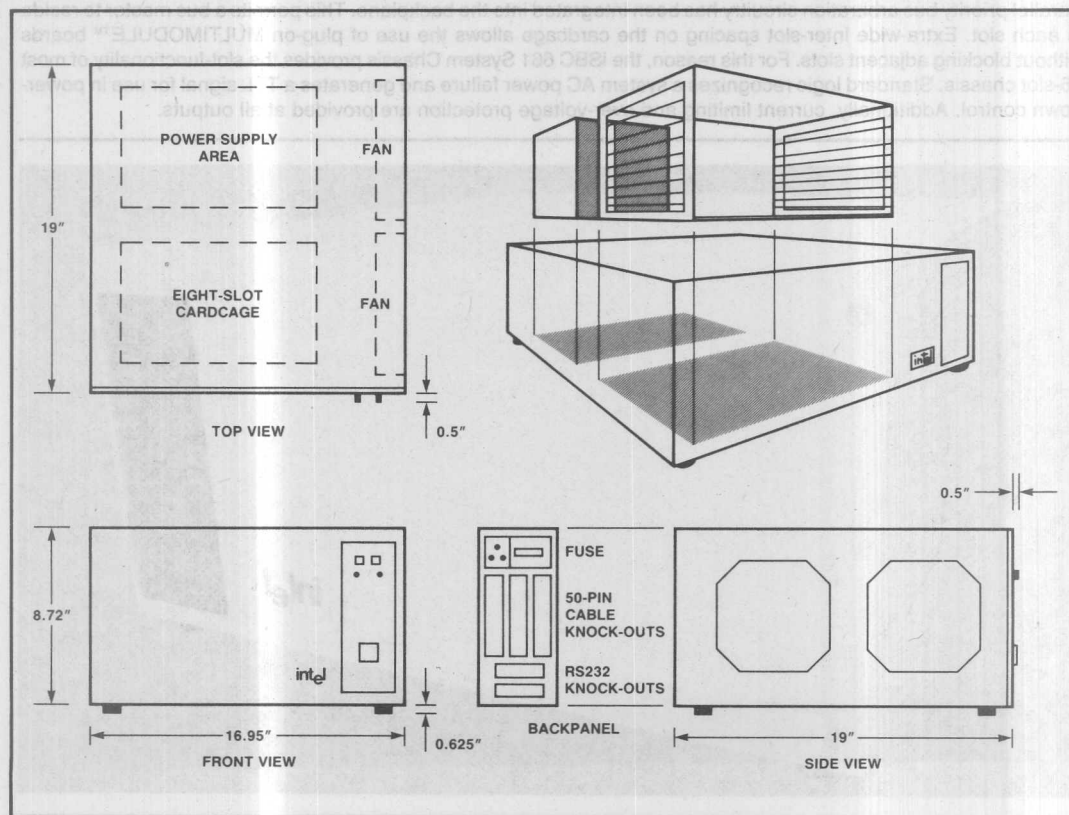


Figure 1. iSBC® 661 System Chassis Dimensions

power, power fail warning, and remote sensing of +5 volts. Its electrical and operational parameters are listed under Specifications.

The cardage of the iSBC 661 System Chassis implements a user-changeable parallel priority bus arbitration scheme by using plug-in jumper connections. Six different priority schemes are allowed, each scheme fixing the priority of the eight MULTIBUS board slots. Bus con-

tention among eight bus-masters in a multiprocessing environment can be managed using this approach.

Noise minimizing ground traces are strategically interleaved between signal and address lines on the system bus. This provides the enhanced noise immunity and minimized signal-to-signal coupling which is particularly important in high speed, high board count microcomputer systems.

SPECIFICATIONS

Electrical Parameters

OUTPUT POWER

Table 1. Output Power Levels iSBC® 661-1

Voltage	Output Current (max.)	Current Limits (amps)	Over-Voltage Protection
+12V	4.5A	4.7-6.8	15V ± 1V
+5V	30.0A	31.5-45.0	6.2V ± 0.4V
-5V	1.75A	1.8-3.2	-6.2V ± 0.4V
-12V	1.75A	1.8-3.2	-15V ± 1V

OPERATIONAL PARAMETERS

Input AC Voltage — 100/120/220/240 VAC ± 10%
(User selects via external switch)

Power-Fail Indication and Hold-Up Time (triggered at 90% of VAC in) — TTL O.C. High 3 msec. (min.)

Output Ripple and Noise — 1% Peak-to-Peak output nominal (DC to 0.5 MHz)

Operational Temperature — 0°C to 50°C

Storage Temperature — -40°C to 70°C

Operational Humidity — 10% to 85% relative, non-condensing

Remote Sensing — Provided for +5 VDC

Output Transient Response — 50 µsec or less for ± 50% load change

PHYSICAL CHARACTERISTICS

Width — 16.95 inches (43.05 cm)

Height — 8.72 inches (22.2 cm)

Depth — 19.00 inches (48.3 cm)

Weight — 41 pounds (21 kg)

Shipping Weight (approx.) — 50 pounds (25 kg)

Equipment Supplied

iSBC® 661-1 — Eight-slot MULTIBUS system chassis chassis with parallel priority arbitration circuitry and 230 watt linear power supply

REFERENCE MANUALS (Not included: order separately)

145340-001 — iSBC 661 System Chassis Hardware Reference Manual

9800803-03 — iSBC 640 Power Supply Hardware Reference Manual

144261-002 — iSBC 608/618 Cardage Hardware Reference Manual

Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department.

In North America: Intel Corp. Literature Department
3065 Bowers Ave.
Santa Clara, California 95051
Phone: (408) 987-8080

In Europe: Intel Corp. S.A. Literature Department
Rue du Moulin A Papier 51
Boite 1
B-1160 Brussels, Belgium
Phone: 322-661-07-11

In the Orient: Intel Corp. Literature Department
5-6 Tokodai, Toyosato-cho
Tsukuba-gun, Ibaragi-ken 300-26
Japan
Phone: 81-29747-8591

User Supplied Options

Compatible Rack-Mount Slides — Chassis Trak, Inc.,
P.O. Box 39100, Indianapolis, IN 46239; Part No.
C 300 S 122

ORDERING INFORMATION

Part Number Description

SBC 6611 Eight-slot MULTIBUS system
chassis with parallel priority
arbitration circuitry and 230 watt
Linear Power Supply

Depth — 19.00 inches (48.3 cm)
Weight — 41 pounds (19 kg)
Shipping Weight (approx.) — 58 pounds (25 kg)

Equipment Supplied
iSBC® 661-1 — Eight-slot MULTIBUS system chassis
chassis with parallel priority arbitration circuitry and 230
watt linear power supply

REFERENCE MANUALS (not included; order separately)
145340-001 — iSBC 661 System Chassis Hardware
Reference Manual
560060-03 — iSBC 640 Power Supply Hardware
Reference Manual
145261-002 — iSBC 608/613 Cardcage Hardware
Reference Manual
Reference manuals may be ordered from any Intel sales
representative, distributor or from Intel Literature
Department.

In North America: Intel Corp. Literature Department
3065 Bore Ave.
Santa Clara, California 95051
Phone: (408) 987-9080
In Europe: Intel Corp. E.A. Literature
Department
Rue du Marais A Papiet 51
Belle 1
B-1180 Brussels, Belgium
Phone: 323-661-07-11
In the Orient: Intel Corp. Literature Department
5-6 Tokodai, Toyokocho-cho
Tatsukawa, Ibaraki-ken 300-26
Japan
Phone: 81-28747-6591

power, power fail warning, and remote sensing of +5
voltage. Electrical and mechanical parameters are listed
under Specifications.

The cardcage of the iSBC 661 System Chassis imple-
ments a novel architecture, controlled directly by address
scheme by using plug-in jumper connections. Six dif-
ferent priority schemes are allowed, each scheme being
the priority of the eight MULTIBUS board slots. Bus con-

SPECIFICATIONS

Electrical Parameters

OUTPUT POWER

Table 1. Output Power Levels iSBC® 661-1

Voltage	Output Current (max.)	Current Limits (amps)	Over-Voltage Protection
+5V	4.5A	4.7-5.8	15V ± 1V
+12V	30.0A	31.5-45.0	8.2V ± 0.4V
-5V	1.75A	1.8-3.2	-8.2V ± 0.4V
-12V	1.75A	1.8-3.2	-18V ± 1V

OPERATIONAL PARAMETERS

Input AC Voltage — 100V/200V/240 VAC ± 10%
(User selects via external switch)
Power-Fail Indication and Hold-Up Time (1) triggered
at 90% of VAC in — TTL O.C. High 3 msec. (min.)
Output Ripple and Noise — 1% Peak-to-Peak output
current (DC to 0.5 MHz)

Operational Temperature — 0°C to 50°C
Storage Temperature — -40°C to 70°C
Operational Humidity — 10% to 85% relative
non-condensing
Remote Sensing — provided for +5 VDC
Quiet Transient Response — 50 nano or less for
± 50mV load change

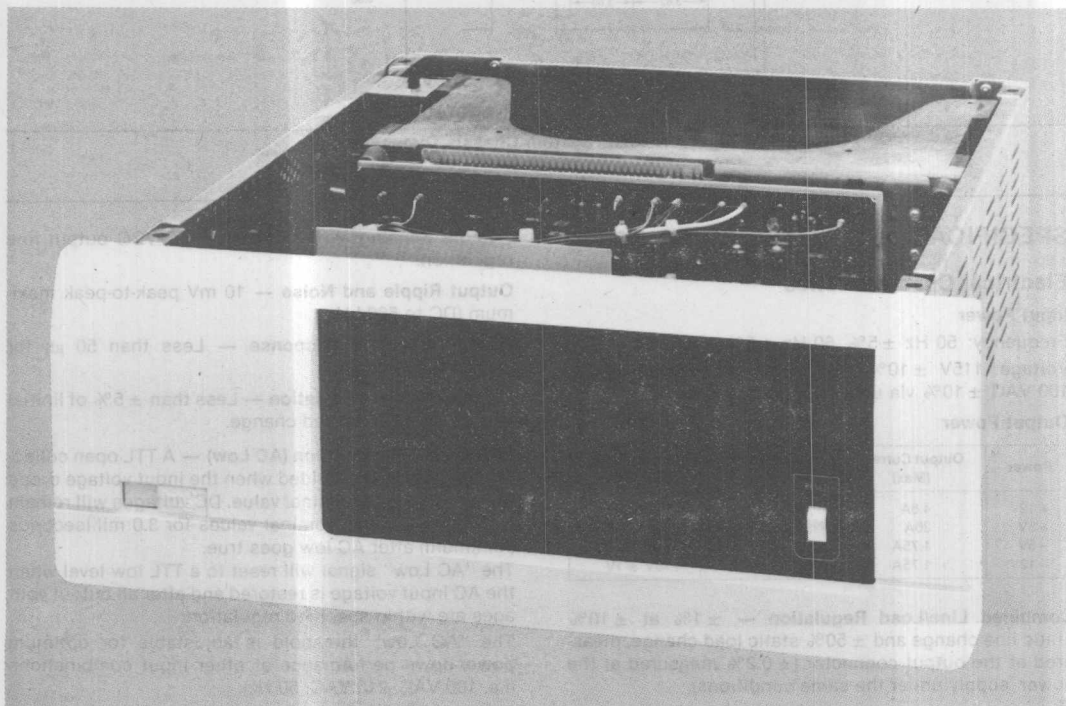
PHYSICAL CHARACTERISTICS

Width — 16.95 inches (43.05 cm)
Height — 8.12 inches (20.6 cm)

SYSTEM CHASSIS

- Eight-slot cardcage and backplane for iSBC computers and expansion boards
- Heavy duty power supply with all standard iSBC voltages
- Compatible with all Intel single board computers
- Forced-air cooling
- Attractive, versatile pop-off front panel
- 19-inch wide rack mountable chassis
- Horizontal board mounting for compactness
- 110/220V, 50/60 Hz operation

The iSBC 660 System Chassis is an attractive, 7-inch high system chassis designed for use with Intel OEM computers. It has eight slots for single board computers, memory, I/O, or other expansion modules. The iSBC 660 is ideal for applications requiring multiple board solutions. DC power output is provided at +12V, +5V, -12V, and -5V levels. The current capabilities of each of these output levels have been chosen to provide power over a 0°C to 50°C temperature range for the majority of applications requiring combinations of computers, memories, peripherals, and other I/O capabilities. Current limiting and over-voltage protection is provided at all outputs. Standard logic recognizes a system AC power failure and generates a TTL signal for use in power-down control. For user convenience, a reset switch is provided on the front panel. The reset signal generated and sent to the system bus can be used for external system control.



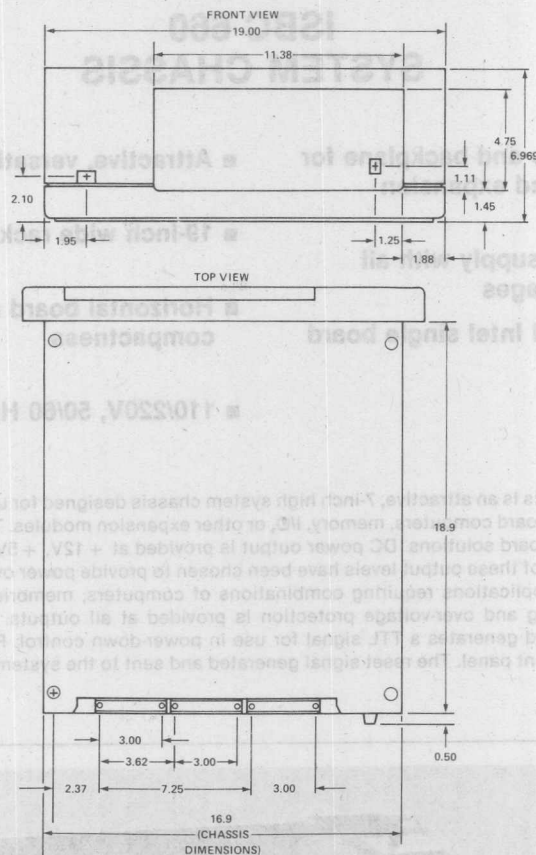


Figure 1. ISBC System Chassis Dimensions

SPECIFICATIONS

Electrical Characteristics

Input Power

Frequency: 50 Hz \pm 5%, 60 Hz \pm 5%

Voltage: 115V \pm 10%, 230V \pm 10%, 215 VAC \pm 10%,
100 VAC \pm 10% via user configured wiring options

Output Power

Power	Output Current (Max)	Current Limit (Amps)	Over-Voltage Protection
+12V	4.5A	5.4	15V \pm 1V
+5V	30A	3.6	6.2V \pm 0.4V
-5V	1.75A	2.1	-6.2V \pm 0.4V
-12V	1.75A	2.1	-15V \pm 1V

Combined Line/Load Regulation — \pm 1% at \pm 10% static line change and \pm 50% static load change, measured at the output connector (\pm 0.2% measured at the power supply under the same conditions).

Remote Sensing — Provided for +5 VDC output line regulation.

Output Ripple and Noise — 10 mV peak-to-peak maximum (DC to 500 kHz).

Output Transient Response — Less than 50 μ s for \pm 50% load change.

Output Transient Deviation — Less than \pm 5% of initial voltage for \pm 50% load change.

Power Failure Indication (AC Low) — A TTL open collector high signal is provided when the input voltage drops below 90% of its nominal value. DC voltages will remain within 5% of their nominal values for 3.0 milliseconds (minimum) after AC low goes true.

The "AC Low" signal will reset to a TTL low level when the AC input voltage is restored and after all output voltages are within specified regulation.

The "AC Low" threshold is adjustable for optimum power-down performance at other input combinations (i.e. 100 VAC, 215 VAC; 50 Hz).

iSBC 660

Humidity — Up to 90% relative, non-condensing

Physical Characteristics

Height — 7 in. (17.8 cm)

Width

At Front Panel: 19 in. (48.3 cm)

Behind Front Panel: 17 in. (43.2 cm)

Depth — 20 in. (50.8 cm) with all protrusions

Environmental Characteristics

Temperature

Operating: 0°C to 50°C

Non-Operating: - 40°C to + 85°C

Equipment Supplied

iSBC 660 System Chassis with iSBC 640 Power Supply, iSBC 604/614 Cardcage/Backplane, dual fans, pop-off front panel

Connector pack with RS232C cable (terminal/modem interface to single board computers), two 50-pin parallel

I/O connectors for single board computers

Schematics for cardcage/backplane, chassis

Outline drawing

Reference Manuals

9800505A — iSBC 660 Hardware Reference Manual (NOT SUPPLIED)

9800505 — iSBC 660 System Chassis Hardware Reference Manual (NOT SUPPLIED)

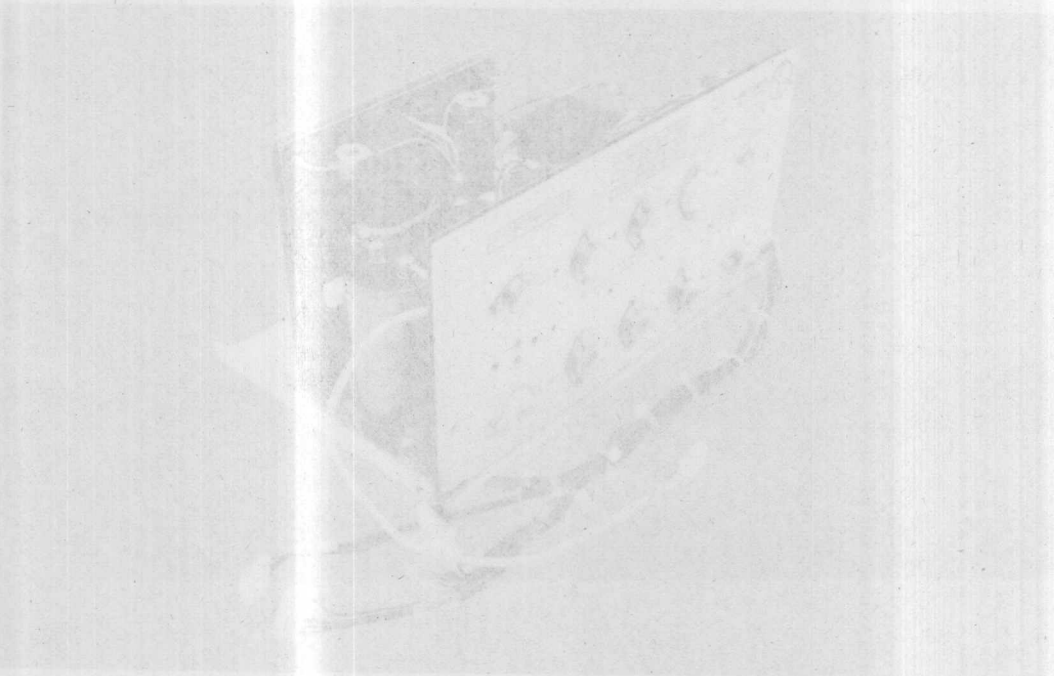
9800803 — iSBC 640 Power Supply Hardware Reference Manual (NOT SUPPLIED)

9800708 — iSBC 604/614 Cardcage Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 660	System Chassis

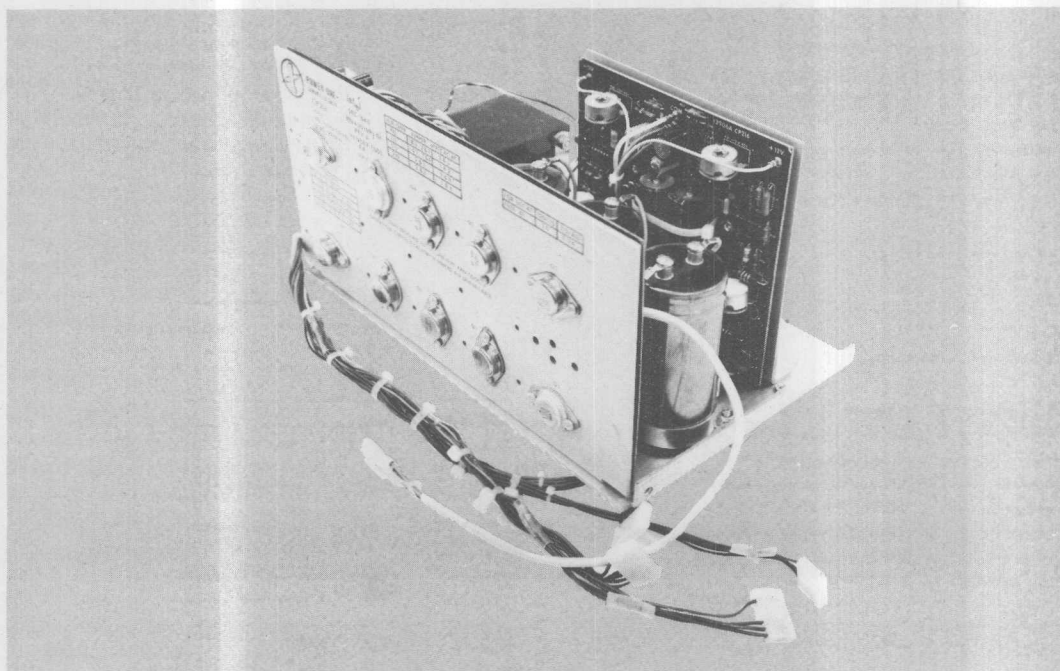




iSBC 640 POWER SUPPLY

- $\pm 5V$ and $\pm 12V$ iSBC 80/86 power
- Sufficient power for 8-12 MULTIBUS computer, memory, and peripheral boards
- Current limiting and overvoltage protection on all outputs
- "AC low" power failure TTL logic level output provided for system power-down control
- Compact single chassis/slide rail mounts in iCS 80 Industrial Chassis or OEM environments
- DC power cables and connectors mate directly to iSBC 604 Modular Cardcage/Backplane assembly
- 100, 115, 215, and 230V AC operation
- 50 Hz or 60 Hz input

The iSBC 640 Power Supply provides low cost, off-the-shelf, single chassis power generation for OEM and industrial system products using Intel single board computers. The iSBC 640 supply provides regulated DC output power at +12V, +5V, -5V and -12V levels. The current capabilities of each of these output levels have been chosen to provide power over a 0°C to +55°C temperature range for one fully loaded Intel single board computer, plus residual capability for most combinations of up to eleven iSBC memory, I/O, or combination expansion boards. Current limiting and overvoltage protection is provided on all outputs. Access for AC input is provided via a standard 4-pin keyed connector. DC output power levels are provided on cables with keyed connectors directly compatible with the iSBC 604/614 Modular Backplane/Cardcage assemblies. The iSBC 640 supply includes logic whose purpose is to sense system AC power failure and generate a TTL signal for clean system power-down control.



SPECIFICATIONS

Electrical Characteristics

Input Power

Frequency: 50 Hz \pm 5%, 60 Hz \pm 5%

Voltage: 115V \pm 10%, 230V \pm 10%, 215VAC \pm 10%,

100VAC \pm 10%

Via user configured wiring options

Output Power

Nominal Voltage	Current (Amps)(Max)	Current Limit Range (Amps)	Short Circuit (Amps)(Max)	Overvoltage Protection
+ 12V	4.5A	4.7- 6.8	2.3	15V \pm 1V
+ 5V	30A	31.5- 45.0	15.0	6.2V \pm 0.4V
- 5V	1.75A	1.8- 3.2	0.9	- 6.2V \pm 0.4V
- 12V	1.75A	1.8- 3.2	0.9	- 15V \pm 1V

Combined Line/Load Regulation — \pm 1% at \pm 10% static line change and \pm 50% static load change, measured at the output connector (\pm 0.2% measured at the power supply under the same conditions).

Remote Sensing — Provided for +5 VDC output line regulation.

Output Ripple and Noise — 10 mV peak-to-peak maximum (DC to 500 KHz)

Output Transient Response — Less than 50 μ sec for \pm 50% load change.

Output Transient Deviation — Less than \pm 10% of initial voltage for \pm 50% load change.

Power Failure Indication (AC Low) — A TTL open collector high signal is provided when the input voltage drops below 90% of its nominal value. DC voltages will remain within 5% of their nominal values for 3.0 milliseconds (minimum, 7.5 ms typical) after AC Low goes true.

The "AC Low" signal will reset to a TTL low level when the AC input voltage is restored and after all output voltages are within specified regulation.

The "AC Low" threshold is adjustable for optimum powerdown performance at other input combinations (i.e. 100 VAC, 215 VAC, 50 Hz).

Mating Connectors¹

AC Input

Housing	Molex	03-09-2042 or equivalent
Pin	Molex	02-09-2118 or equivalent (18 to 22 gauge wire)

DC Output²

Housing	Molex	26-03-3071
	Amp	3-87025-3
Pins	Molex	08-50-0187 or 08-50-0189
	Amp	87023-1
Key	Molex	15-04-9209
	Amp	87116-2

Compatible with Molex 09-66-1071 Header

Notes

1. Pins from given vendor may only be used with connectors from the same vendor.
2. iSBC 640 DC output connectors are directly compatible with input power connectors on iSBC 604 Modular Cardcage/Backplane assembly. Four connectors are provided.

Physical Characteristics

Height — 6.66 in. max. (16.92 cm)

Width — 8.19 in. max. (20.80 cm)

Depth — 12.65 in. max. (32.12 cm)

Weight — 30 lbs. max (13.63 kg)

Environmental Characteristics

Temperature — 0°C to 55°C with 55 CFM moving air

Non-Operating — - 40°C to + 85°C

Equipment Supplied

iSBC 640 Power Supply with AC and DC cables with keyed connectors.

Reference Manuals

9800803 — iSBC 640 Power Supply Hardware Reference Manual (includes schematic and assembly drawings) (NOT SUPPLIED)

9800798 — iCS 80 Systems Site Planning and Installation Manual (for installation of iSBC 640 supply into iCS 80 Industrial Chassis) (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 640	Power Supply

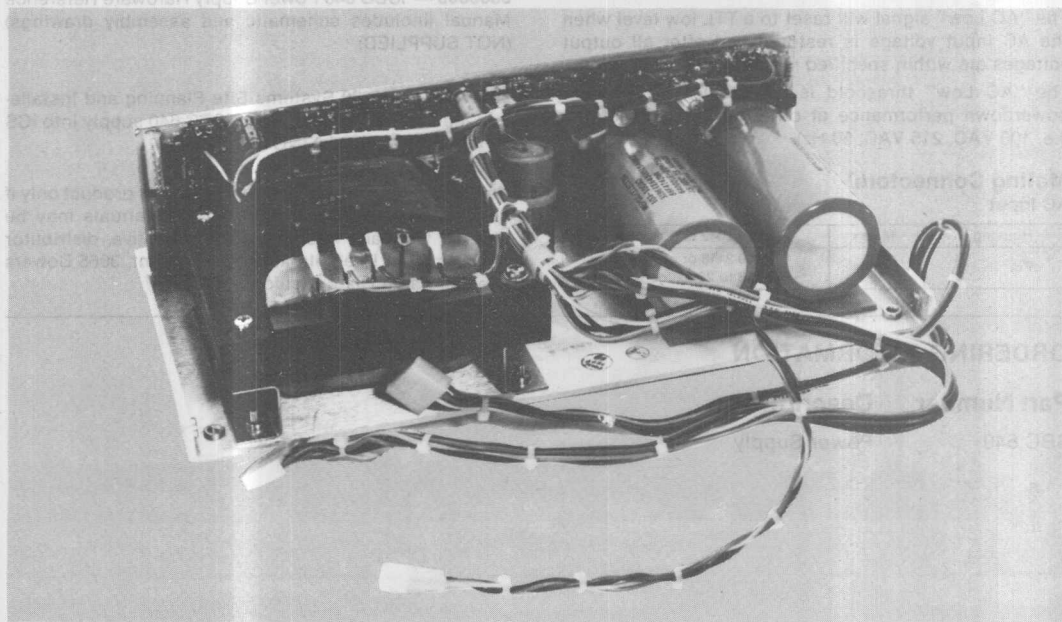


iSBC 635 POWER SUPPLY

- Compact single chassis
- $\pm 5V$ and $\pm 12V$ iSBC 80 and iSBC 86 system power
- Sufficient power for one fully loaded Intel single board computer plus residual power for up to three Intel iSBC expansion boards
- Current limiting and overvoltage protection on all outputs

- DC power cables and connectors mate directly to iSBC 604 Modular Cardcage/Backplane assembly
- "AC low" power failure TTL logic level output provided for system power-down control
- 100V, 115V, 215V, and 230V AC operation
- 50 Hz or 60 Hz input

The iSBC 635 Power Supply provides low cost, off-the-shelf, single chassis power generation for OEM products using Intel single board computers. The iSBC 635 supply provides regulated DC output power at $+12V$, $+5V$, $-12V$, and $-12V$ levels. The current capabilities of each of these output levels have been chosen to provide power over a $0^{\circ}C$ to $+55^{\circ}C$ temperature range for one Intel single board computer fully loaded with I/O line terminators and drivers and EPROMs, plus residual capability for most combinations of up to three iSBC memory, I/O or combination expansion boards. Current limiting and overvoltage protection is provided on all outputs. Access for AC input is provided via a standard 4-pin keyed connector. DC output power levels are provided on cables with keyed connectors directly compatible with the iSBC 604 Modular Cardcage/Backplane assembly. The iSBC 635 supply includes logic whose purpose is to sense system AC power failure and generate a TTL signal for clean system power-down control.



ISBC 635

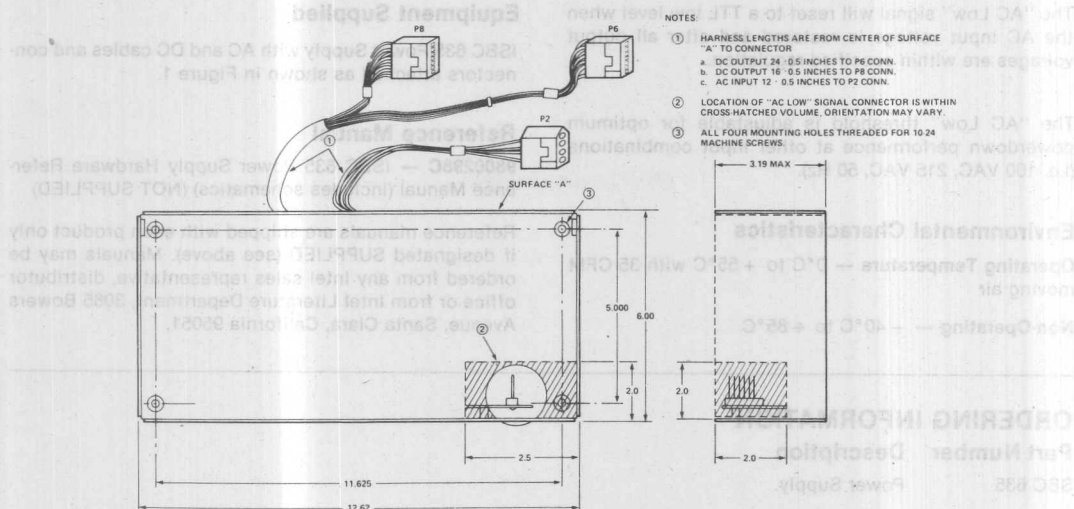


Figure 1. ISBC 635 Mounting Information

SPECIFICATIONS

Mating Connectors¹

AC Input

Connector	Molex	03-09-1042 or equivalent
Pin	Molex	02-09-1118 or equivalent (18 to 22 gauge wire)

DC Output²

Header	Molex	09-66-1071
	AMP	87194-6

"AC Low" Control

Connector	Molex	09-50-7071
	AMP	87159-7
Polarizing key	Molex	15-04-0219
	AMP	87116-2
Pin	Molex	08-50-0106 (18 to 22 gauge wire)
	AMP	87023-1 (18 to 22 gauge wire)

Notes

1. Pins from a given vendor may only be used with connectors from the same vendor.
2. ISBC 635 DC output connectors are directly compatible with power input power connectors on ISBC 604 Modular Cardcage/Backplane assembly. Two connectors are provided.

Physical Characteristics

Height — 3.19 in. max (8.11 cm)
Width — 6.03 in. max (15.32 cm)
Depth — 12.65 in. max (32.12 cm)
Weight — 13 lb (5.90 kgm)

Electrical Characteristics

Input Power — Frequency: 47 - 63 Hz. Voltage (Nominal) (Single Phase): 100, 115, 215, or 230 VAC +10%

Output Power:

Nominal Voltage	Current (AMPS)(MAX)	Current Limit Range (AMPS)	Max Short Circuit (AMPS)	Over-Voltage Protection
+12	2.0	2.1-3.0	1.0 (Foldback)	+14 to +16 V
+ 5	14.0	14.7-21.0	7.0 (Foldback)	+5.8 to +6.6 V
- 5	0.9	0.9-1.4	1.4	-5.8 to -6.6 V
-12	0.8	0.8-1.2	1.2	-14 to -16 V

Combined Line/Load Regulation — $\pm 1\%$ at $\pm 10\%$ static line change and $\pm 50\%$ static load change, measured at the output connector ($\pm 0.2\%$ measured at the power supply under the same conditions).

Remote Sensing — Provided for +5VDC output line regulation.

Output Ripple and Noise — 10 mV peak-to-peak maximum (DC to 500 KHz)

Output Transient Response — Less than 50 μ sec for $\pm 50\%$ load change

Output Transient Deviation — Less than $\pm 5\%$ of initial voltage for $\pm 50\%$ load change.

Power Failure Indication (AC Low) — A TTL open collector high signal is provided when the input voltage drops below 90% of its nominal value. DC voltages will remain within 5% of their nominal values for 3.0 milliseconds (minimum) after AC low goes true.

iSBC 635

The "AC Low" signal will reset to a TTL low level when the AC input voltage is restored and after all output voltages are within specified regulation.

The "AC Low" threshold is adjustable for optimum powerdown performance at other input combinations (i.e. 100 VAC, 215 VAC, 50 Hz).

Environmental Characteristics

Operating Temperature — 0°C to +55°C with 35 CFM moving air

Non-Operating — -40°C to $+85^{\circ}\text{C}$

ORDERING INFORMATION

Part Number	Description
-------------	-------------

SBC 635 Power Supply

Equipment Supplied

iSBC 635 Power Supply with AC and DC cables and connectors attached as shown in Figure 1.

Reference Manual

9800298C — iSBC 635 Power Supply Hardware Reference Manual (includes schematics) (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

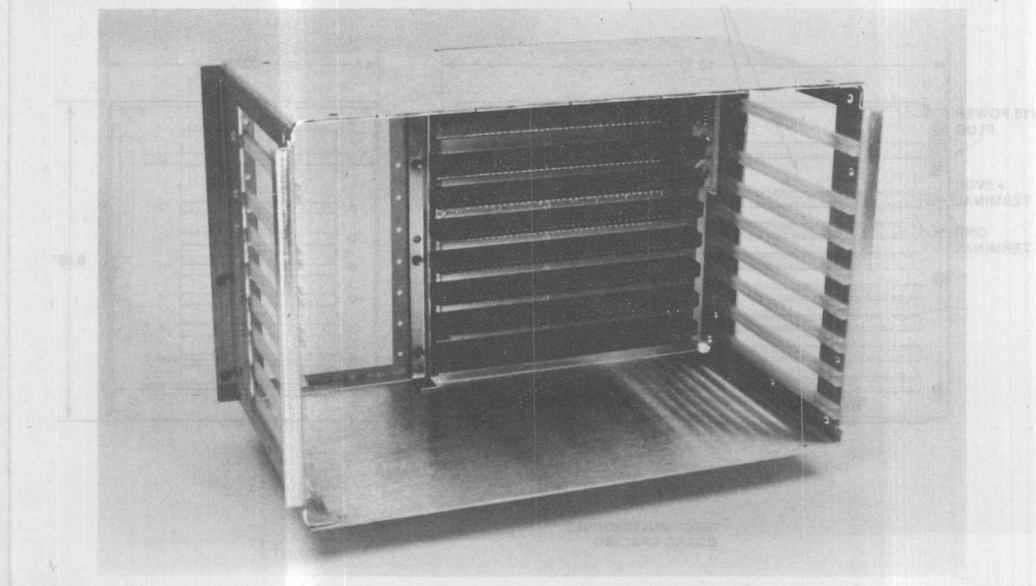


iSBC® 608/618 CARDCAGES

- Houses eight MULTIBUS® iSBC® boards in an aluminum package
- Board-to-board clearance for iSBC® MULTIMODULE™ boards on all slots
- Board-to-board clearance for iSBX™ MULTIMODULE™ boards on two slots
- Parallel priority circuitry for up to eight Multimaster iSBC® boards
- Enhanced bus noise immunity for high speed systems
- Plug on iSBC® 618 unit for up to sixteen board systems
- NEMA-type backwall or 19-inch rack mount hardware included
- Signal line termination circuitry on iSBC® 608 Cardcage

Intel's iSBC 608/618 Cardcages are matched to the latest generation of iSBC/iSBX boards which mount in the MULTIBUS system bus. These products provide several features which make them the industry's leading price/performance cardcage product. MULTIMODULE board clearance, parallel priority circuitry, enhanced backplane noise immunity, and precision fit card guides are a few of the distinctions which make this the industry's better product.

The iSBC 608 Cardcage is the base unit, housing up to eight iSBC boards and their MULTIMODULE boards. Additionally, this base unit provides mounting hardware and fan mounting bracketry. The iSBC 618 is the expansion unit, providing eight additional iSBC board slots to the iSBC 608 Cardcage for a total of sixteen board slots which can be NEMA-type backwall or 19-inch rack mounted. This is accomplished with the mounting hardware of the iSBC 608 Cardcage. The iSBC 618 expansion unit also provides fan mounting bracketry.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Mechanical Aspects

The iSBC 608/618 Cardcages provide housing and a MULTIBUS system bus for up to sixteen single board computers and their MULTIMODULE boards. The iSBC 608 unit and iSBC 618 unit offer board-to-board clearance (0.8 inches or greater) on all eight slots for iSBC MULTIMODULE boards. Two slots provide clearance (1.2 inches or greater) for iSBX MULTIMODULE boards as shown in Figure 1. Each cardcage includes precision fitted nylon card-guides for secure board fit and accurate MULTIBUS board pin alignment. Fan mounting bracketry is also included with each cardcage. This bracketry allows the mounting of several industry standard fans. The iSBC 608 Cardcage base unit includes aluminum mounting hardware for NEMA-type backwall mounting, or anchoring a sixteen slot iSBC 608/618 combination in a standard 19-inch rack.

Electrical Aspects

The iSBC 608/618 Cardcages implement a parallel priority resolution scheme by using plug-in jumper

connections. There are six different priority schemes allowed, each requiring a different jumper configuration. In systems where an iSBC 618 Cardcage is attached to the base unit, the base unit will have lower priority overall. That is, master boards in the iSBC 608 base unit may gain control of the MULTIBUS lines only when no boards in the iSBC 618 expansion unit are asserting the bus request (BREQ/) signal.

Noise-minimizing ground traces are strategically interleaved between signal and address lines on these backplanes. This provides the enhanced noise immunity and minimized signal-to-signal coupling which is important in high speed, high board count microcomputer systems.

The iSBC 608/618 Cardcages provide power connector lug bolts for +5 VDC and ground. The lug bolts, compared to other power connection methods, help transfer higher amounts of current. Other voltages (± 12 VDC, -10 VDC) are connected via a mating power connector plug as shown in Figure 2.

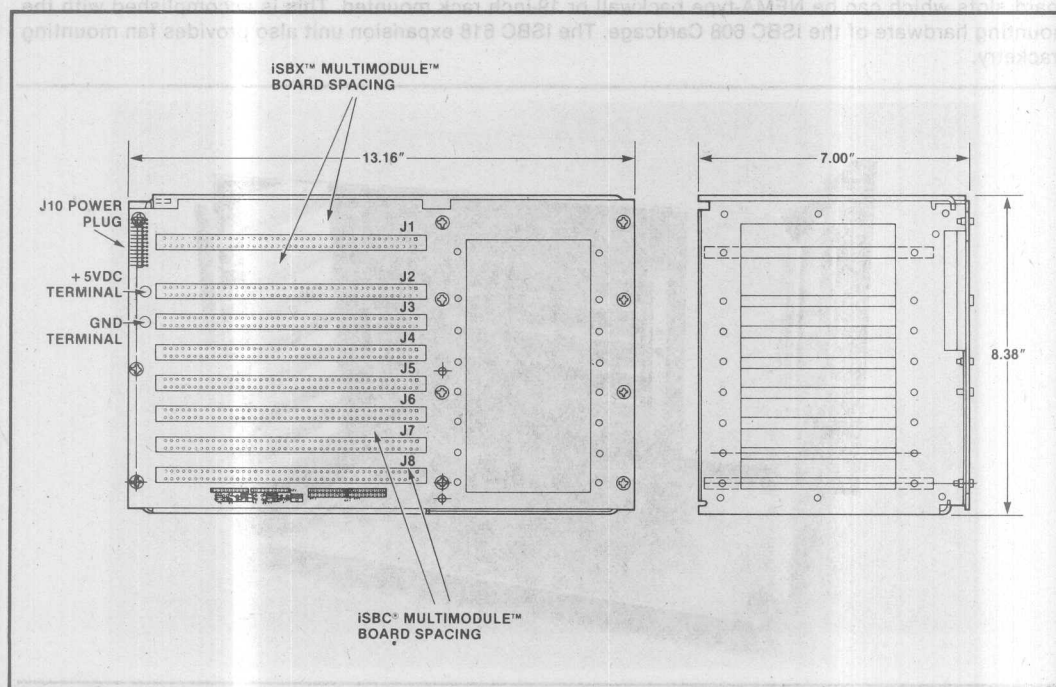


Figure 1. iSBC® 608/618 Cardcages Dimensions

SPECIFICATIONS

Bus Lines

All MULTIBUS (IEEE 796) system bus address and command lines are bussed to each of the eight MULTIBUS connectors on the backplane. Ground traces are interleaved among these signal lines and bussed to the backplane edge connector for interconnection of the ISBC 608 and ISBC 618 backplane.

Power Connectors

Ground (0V), +5V, -5V, +12V, -12V, -10V power supply header stakes and power lug bolts are provided on the ISBC 608/618 Cardcages as shown in Figure 2.

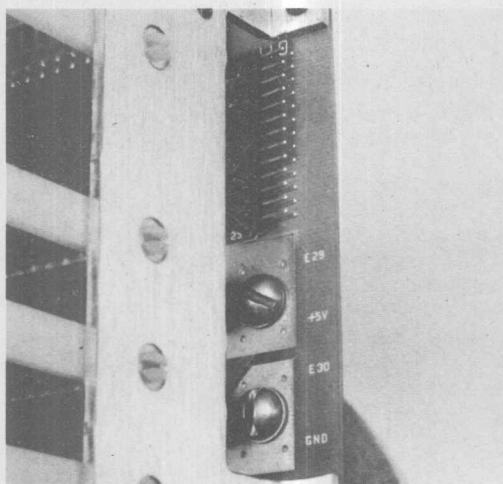


Figure 2. Power Header Stakes and Lugs

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Storage Temperature — -40°C to 85°C

Humidity — 50% to 95% non-condensing at 25°C to 40°C

Vibration and Shock — 2G max. through 50 Hz

Physical Characteristics

SLOT-TO-SLOT DIMENSIONS (See Figure 1)

Top-J1 — 1.200 in (to center)

J1-J2 — 1.300 in (center to center)

J8-Bottom — 0.700 in (to center)

All Others — 0.800 (center to center)

PHYSICAL DIMENSIONS

Height — 8.38 in (21.29 cm)

Length — 13.16 in (33.43 cm)

Width — 7.50 in (19.05 cm)

Weight — 3.50 lbs (1.59 kg)

Shipping Weight — 5.75 lbs (2.61 kg)

Equipment Supplied

ISBC® 608 BASE UNIT

Eight-Slots — Two at greater than 1.2 inches; six at 0.8 inches

Male Backplane Connector — For expansion with ISBC 618 cardcage

Parallel Priority Circuitry — Eight slots are configurable via the use of jumper stakes. Six priority schemes allowed

Construction Materials —

Aluminum card housing

Nylon card guides

Power connector header stakes and lug bolts

ISBC® 618 EXPANSION UNIT

Eight-Slots — Two at greater than 1.2 inches; six at 0.8 inches

Female Backplane Connector — For expansion to ISBC 608 base unit

Parallel Priority Circuitry — Eight slots are configurable via the use of jumper stakes. Six priority schemes allowed

Construction Materials —

Aluminum card housing

Nylon card guides

Power connector header stakes and lug bolts

User-Supplied Equipment

MATING POWER CONNECTORS

Vendor	Part Number
3M	3399-6026
Ansley	609-2600M
Berg	65485-009

MOUNTABLE FANS

Vendor	Part Number
Rotron	SU2A1-028267
Torin	TA300-A30473-10
Pamotor	8506D

Reference Manual

144261-001 — iSBC 608/618 Cardcages Hardware Reference Manual (order separately)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051

ORDERING INFORMATION

Part Number	Description
SBC 608	Cardcage (base unit)
SBC 618	Cardcage (expansion unit for iSBC 608)

All MULTIBUS (IEEE 796) system bus address and command lines are located to each of the eight multibus connectors on the backplane. Ground traces are interleaved among these signal lines and placed to the backplane edge connector for interconnection of the iSBC 608 and iSBC 618 backplane.

Power Connectors
Ground (0V), +5V, -5V, +12V, -12V, -10V power supply header stakes and power lug bolts are provided on the iSBC 608/618 Cardcages as shown in Figure 2.

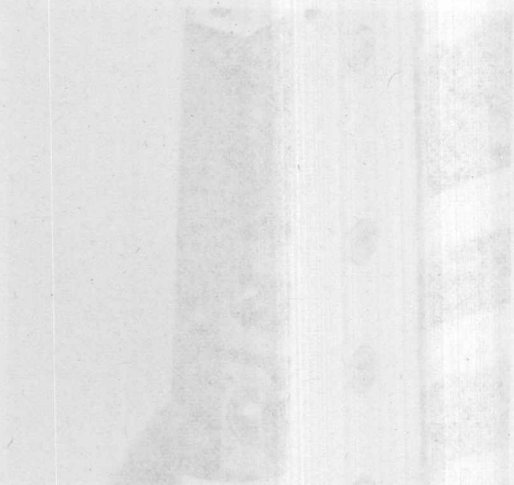


Figure 2. Power Header Stakes and Lugs

Environmental Characteristics
Operating Temperature — 0°C to 55°C
Storage Temperature — -40°C to 85°C
Humidity — 30% to 95% non-condensing at 25°C to 40°C

Vibration and Shock — 5G max. through 80 Hz
Physical Characteristics

SLOT-TO-SLOT DIMENSIONS (See Figure 1)

Top — 1.300 in (to center)
Bottom — 1.300 in (to center)
Left — 0.700 in (to center)
Right — 0.700 in (to center)

USER-SUPPLIED EQUIPMENT

Vendor	Part Number
3M	3399-8026
Analog	808-3000M
Borg	65463-001

MOUNTABLE FANS

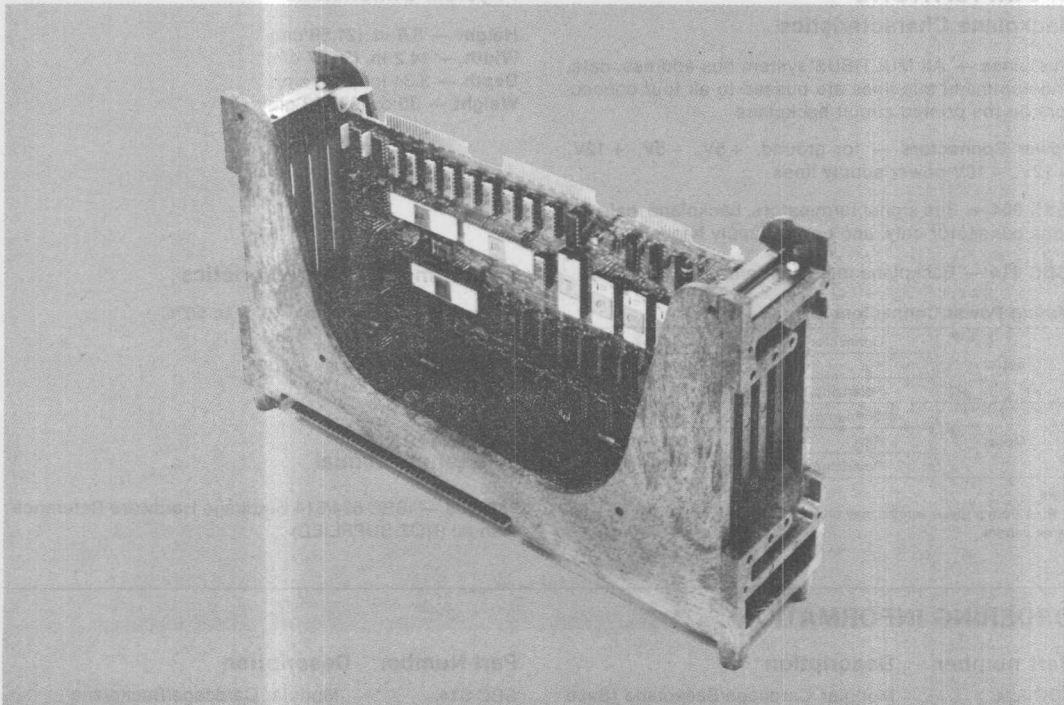
Vendor	Part Number
Boyer	803A1-02826
Form	7A300-A3047-10
Panasonic	5508D



iSBC 604/614 or (pSBC 604/614*) MODULAR CARDCAGE/BACKPLANE

- Interconnection on MULTIBUS system bus and housing for up to four Intel iSBC boards
- Strong cardcage structure helps protect installed iSBC single board computers and expansion boards against warping and physical damage
- Connectors allow interconnection of two or more cardcage/backplane assemblies
- Cardcage mounting holes facilitate interconnection of units
- Compatible with 3.5-inch RETMA rack mount increments
- Dual backplane power supply connectors and signal line termination circuits on iSBC 604 Cardcage/Backplane

The iSBC 604 and iSBC 614 Modular Cardcage/Backplane units provide low-cost, off-the-shelf housing for OEM products using two or more Intel single board computers. Each unit interconnects and houses up to four boards. The base unit, the iSBC 604 Cardcage/Backplane, contains a male backplane PC edge connector and bus signal termination circuits, plus power supply connectors. It is suitable for applications requiring a single unit, or may be interconnected with the iSBC 614 Cardcage/Backplane when more than one cardcage/backplane unit is needed. The iSBC 614 Cardcage/Backplane contains both male and female backplane connectors, and may be interconnected with iSBC 604/614 Cardcage/Backplane units. Both units are identical, with the exception of the power connectors and bus signal terminator features. A single unit may be packaged in a 3.5-inch RETMA rack enclosure, and two interconnected units may be packaged in a 7-inch enclosure. The units are mountable in any of three planes.



*Same product, manufactured by Intel Puerto Rico, Inc.

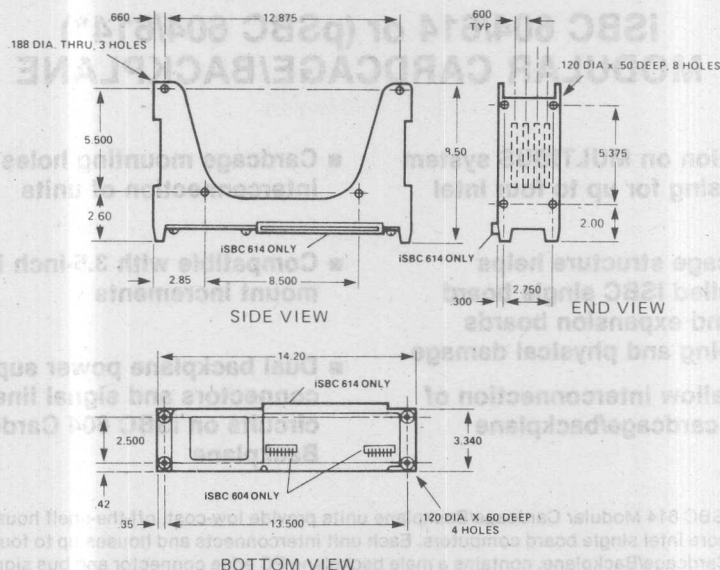


Figure 1. iSBC 604/614 Modular Backplane and Cardcage Dimensions

SPECIFICATIONS

Backplane Characteristics

Bus Lines — All MULTIBUS system bus address, data, and command bus lines are bussed to all four connectors on the printed circuit backplane

Power Connectors — for ground, +5V, -5V, +12V, -12V, -10V power supply lines

iSBC 604 — Bus signal terminators, backplane male PC edge connector only, and power supply headers

iSBC 614 — Backplane male and female connectors

Mating Power Connectors

AMP	Connector	87159-7
	Pin	87023-1
	Polarizing key	87116-2
Molex	Connector	09-50-7071
	Pin	08-50-0106
	Polarizing key	15-04-0219

Note

1. Pins from a given vendor may only be used with connectors from the same vendor.

Physical Dimensions

Height — 8.5 in. (21.59 cm)

Width — 14.2 in. (36.07 cm)

Depth — 3.34 in. (8.48 cm)

Weight — 35 oz (992.23 gm)

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Reference Manual

9800708 — iSBC 604/614 Cardcage Hardware Reference Manual (NOT SUPPLIED)

ORDERING INFORMATION

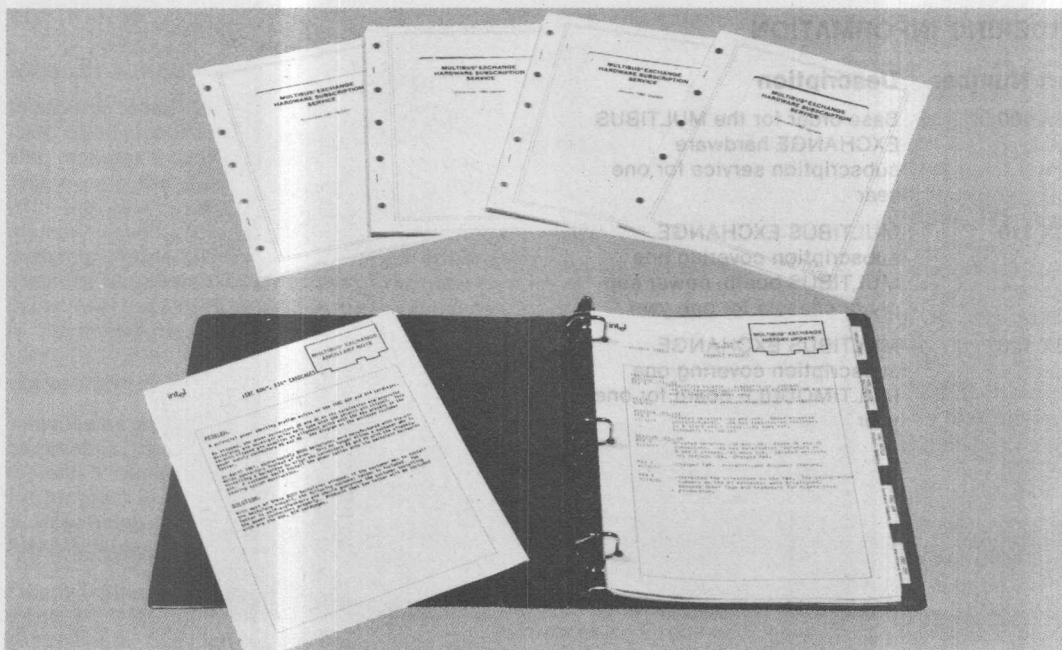
Part number	Description
SBC 604	Modular Cardcage/Backplane (Base Unit)

Part Number	Description
SBC 614	Modular Cardcage/Backplane (Expansion Unit)

IMBX 100/110/120 MULTIBUS® EXCHANGE HARDWARE SUBSCRIPTION SERVICE

- Monthly product updates
- History of Engineering Change Orders
- Customized to the subscriber's selection of Intel® board level products
- Ancillary Notes clarifying and correcting product documentation
- Offers full update alternatives — documentation, parts kit, and factory update
- Timely announcements of technological and product developments
- Available for iSBC,™ iCS,™ and iSBX™ products

The Intel MULTIBUS EXCHANGE hardware subscription service consists of a monthly update publication which provides summary descriptions of Engineering Change Orders that alert and inform subscribers of all the latest developments and/or improvements applicable to their Intel board level products. In addition, the update contains order numbers and prices for optional update parts kits. This timely service allows users to closely track product changes and, by evaluating or implementing these changes, to take advantage of technological and product developments.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

PRODUCTS

The MULTIBUS EXCHANGE hardware subscription service supports Intel manufactured Single Board Computer, Industrial Control Series and MULTIMODULE products. Users can subscribe to any combination of over 75 products.

NEW SUBSCRIBERS

New subscribers receive a history for each subscribed-to product and a handsome notebook in which future updates may be orderly stored.

MONTHLY UPDATES

Each month MULTIBUS EXCHANGE subscribers receive an update containing summaries of Engineering Change Orders and documentation clarification articles relating to each subscribed-to product. Each change description includes the change made, the rationale behind the change, and the user's alternatives concerning updating

his product. Subscribers receive the monthly update even if no changes have occurred to their products.

ORDERS

Orders for the MULTIBUS EXCHANGE are a combination of a base (IMBX 100) order, plus orders for each product to be included (IMBX 110s and/or IMBX 120s). For each MULTIBUS board, power supply, or chassis, an IMBX 110 is ordered. For each MULTIMODULE board, an IMBX 120 is ordered.

CHARGES

For an annual subscription fee, subscribers receive twelve monthly updates. All new subscribers will also receive a history for each subscribed-to product at no extra charge. For additional specified charges, subscribers can purchase the associated documentation and parts kits listed in the MULTIBUS EXCHANGE updates.

ORDERING INFORMATION

Part Number	Description
MBX 100	Base order for the MULTIBUS EXCHANGE hardware subscription service for one year
MBX 110	MULTIBUS EXCHANGE subscription covering one MULTIBUS board, power supply or chassis for one year
MBX 120	MULTIBUS EXCHANGE subscription covering one MULTIMODULE board for one year

*Integrated
Microcomputer Systems*

2

iSXM™ 100

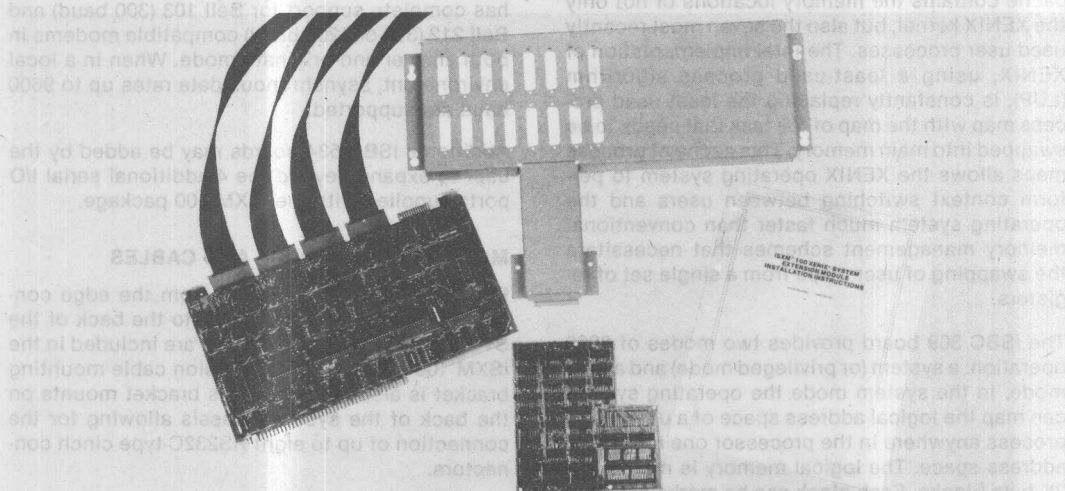
SYSTEM 86/300 SERIES

XENIX* SYSTEM EXTENSION MODULE

- Complete hardware extension to execute XENIX* on Intel® SYSTEM 86/300 Series microcomputers.
- Provides memory management and protection to the iAPX 86 processor family.
- Process map cache designed to provide high performance context switching for user tasks.
- Includes 4 serial connections and can be expanded with additional serial expansion boards.
- Supported by SYSTEM 86/300 Series diagnostics for ease of repair.
- Includes all necessary cables and mounting hardware.
- Simple integration into SYSTEM 86/300 Series microcomputers.

The iSXM 100 XENIX System Expansion Module is designed to extend the hardware capability of iSBC 86/30-based SYSTEM 86/300 Series microcomputers in order to execute the multi-user, XENIX interactive operating system. This field-installed module provides memory management and memory protection optimized for use with XENIX, four additional serial ports, internal cables and cable mounting hardware.

All hardware is fully configured and can be easily installed in the system. An easy-to-follow installation manual as well as all hardware documentation is included with the iSXM 100 package. In addition, as with all hardware and peripherals in the system, the expansion module is totally supported by extensive diagnostic routines. These diagnostics provide a convenient and easy to use interface for diagnosing possible problems and for rapid repair.



* XENIX is a trademark of Microsoft Corporation.

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellex, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

TECHNICAL DESCRIPTION

The iSXM 100 package is composed of several standard modules including the Intel iSBC 309 Memory Management MULTIMODULE board, iSBC 534 4-port Serial Expansion board, cables and mounting hardware. The addition of this kit allows the OEM to execute the XENIX interactive operating system on Intel boards and systems.

Memory Management

The Intel iSBC 309 Memory Management MULTIMODULE board is designed to add memory capability to the Intel iSBC 86/30-based processor board used in SYSTEM 86/300 Series microcomputers. The iSBC 309 board, which operates at either 5 or 8 MHz, plugs directly into the processor socket of the iSBC 86/30 board and it does not require any other interface or power requirements. XENIX takes advantage of these hardware features through the use of a dynamic scatter loading technique. This method of memory management allows programs to be loaded in non-contiguous blocks of RAM for more efficient memory utilization.

The architecture of the iSBC 309 Memory Management board is specially designed to optimize the performance of the XENIX operating system. On the iSBC 309 board there is an eight register set that makes up a process map cache. Through software support in the XENIX operating system, this cache contains the memory locations of not only the XENIX kernel, but also the seven most recently used user processes. The Intel implementation of XENIX, using a least-used process algorithm (LUP), is constantly replacing the least used process map with the map of the task that needs to be swapped into main memory. This cache of process maps allows the XENIX operating system to perform context switching between users and the operating system much faster than conventional memory management schemes that necessitate the swapping of user maps from a single set of registers.

The iSBC 309 board provides two modes of 8086 operation: a system (or privileged mode) and a user mode. In the system mode the operating system can map the logical address space of a user mode process anywhere in the processor one megabyte address space. The logical memory is mapped in 2K byte blocks. Each block can be marked as non-existent, read only, or read/write memory. These operations are performed automatically by the XENIX operating system and are transparent to the user.

In the user mode a process cannot alter the memory map or change the process number. Depending on the XENIX configuration, up to 7 user processes (plus one system process) with independent memory maps can be actively mapped at one time. The ability to support multiple processes reduces the operating system overhead by reducing the time spent loading the memory map.

The XENIX operating system available from Intel takes full advantage of these powerful memory management features available on the iSBC 309 board. For more detailed information on the iSBC 309 Memory Management board refer to the iSBC 309 Memory Management board data sheet, number 210496-001.

Serial I/O Expansion

In addition to the one serial I/O channel resident on the iSBC 86/30 processor board, the iSXM 100 package provides four additional ports through the use of an Intel 534 Serial Expansion board.

The iSBC 534 board is shipped fully configured and ready to install in the SYSTEM 86/300 Series microcomputer. The board interfaces directly to the system through the industry standard (IEEE-P796) MULTIBUS system bus. The four serial ports fully support RS232C (configured) asynchronous communications.

The XENIX operating system available from Intel has complete support for Bell 103 (300 baud) and Bell 212 (300 or 1200 baud) compatible modems in both answer and originate mode. When in a local environment, asynchronous data rates up to 9600 baud are supported.

Additional iSBC 534 boards may be added by the user to expand beyond the 4 additional serial I/O ports supplied with the iSXM 100 package.

MOUNTING HARDWARE AND CABLES

Four RS232C cables for use from the edge connectors of the iSBC 534 board to the back of the SYSTEM 86/300 Series chassis are included in the iSXM 100 package. An expansion cable mounting bracket is also supplied. This bracket mounts on the back of the system chassis allowing for the connection of up to eight RS232C-type cinch connectors.

DIAGNOSTICS

When the Extension Module is used with Intel SYSTEM 86/300 Series microcomputer systems,

there is a two level diagnostic program that supports both the iSBC 309 Memory Management board and the iSBC 534 Serial I/O Expansion board. The first level of diagnostics, the System Confidence Test (SCT) is based in EPROM on the iSBC 86/30 processor board. This test gives a go/no-go condition on all hardware and peripherals present in the system. If a no-go condition exists, then the second level of diagnostics may be executed. The disk-based Systems Diagnostic Test (SDT) runs detailed diagnostics on specific pieces of hardware such as the memory boards, processor or memory management unit.

SPECIFICATIONS

Hardware Supplied

iSBC 309 Memory Management board
iSBC 534 4-port Serial I/O Expansion board
Four card edge to RS232C serial cables
Serial cable mounting bracket

Electrical Requirements

Voltage —	Current Draw (Max)	
	iSBC 309	iSBC 534
+ 5	2A	1.9 A
+ 12	—	420 mA
- 12	—	400 mA

Environmental Characteristics

OPERATING TEMPERATURE

iSXM™ 100 — 0° to 55°C

SYSTEM 86/300 Series Microcomputers —
15° to 35°C

INSTALLATION

The iSXM 100 XENIX System Extension Module is designed to be field installed into any iSBC 86/30-based SYSTEM 86/300 Series microcomputer. An installation guide giving detailed instructions is included with the iSXM 100 package. Total installation time for the hardware kit is estimated to be less than one hour. Complete hardware installation can also be provided by Intel Field Service Engineers for an optional fee. Contact the local Intel Sales Office or Intel distributor for more information on field service installation.

Communication Capabilities

Hardware — iSBC 534 4-port Serial I/O Expansion board

Interface Support — RS232C

Modem Support — 300 baud — Bell 103A or equivalent; 300 baud, 1200 baud — Bell 212A or equivalent

Physical Characteristics

	iSBC 309	iSBC 534
Width —	6.00 in (15.24 cm)	12.00 in (30.48 cm)
Length —	5.75 in (14.61 cm)	6.75 in (17.15 cm)
Height —	1.14 in (2.90 cm)	.50 in (1.27 cm)
iSXM™ 100 Shipping Weight —	10 pounds	

Reference Manuals

144686-001 iSBC 309 User's Guide
9800450-02 iSBC 534 Hardware Reference Manual
144778-001 iSXM 100 XENIX Installation Guide

ORDERING INFORMATION

The iSXM 100 XENIX System Extension Module may be ordered separately as the iSXM 100 module or as part of the SYSTEM 86/300X or 86/380X kit.

The XENIX software package may also be ordered separately from Intel as SD12XNX86H.



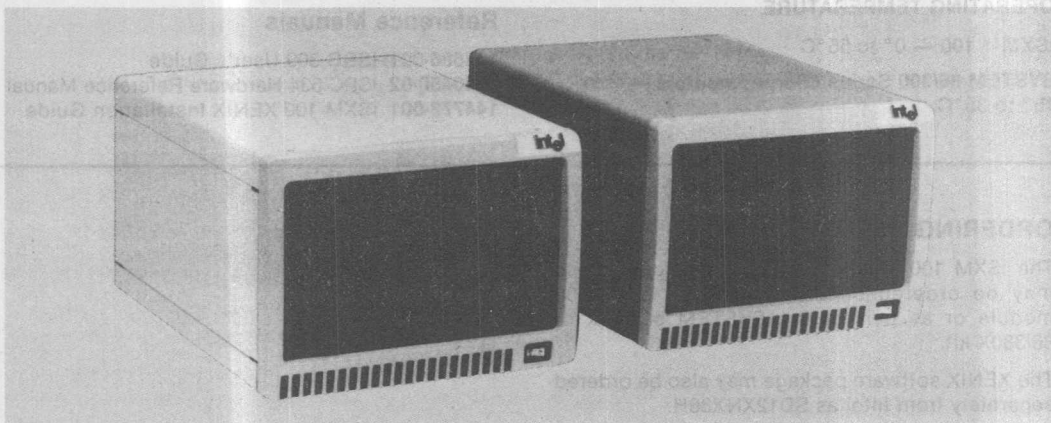
12XM 100



SYSTEM 86/380X MICROCOMPUTER SYSTEM XENIX*

- Industry standard XENIX interactive operating system
- Memory management and protection for the high-performance 8 MHz 16-bit iAPX 86/10 processor
- Standard language support includes "C" and ASSEMBLER-86. Other languages available soon
- Expansion provided for ten MULTIBUS® iSBC® boards and one 8-inch standard peripheral in two desk-top or rack-mount chassis
- Meets the open system requirements of the OEM
- 35MB Winchester and 1MB diskette drive for program, data storage, and back-up
- 384KB of high speed RAM memory for multiple users and processes
- Extensive self-test routines to ensure reliable operation and enhance fault isolation
- Five serial ports included (expandable to nine ports)

The Intel SYSTEM 86/380X Microcomputer System is an integrated, high-performance 16-bit XENIX-based hardware/software package designed for the OEM, yielding all the benefits of the latest advances in VLSI technology and performance without requiring complex system integration. The system, based on standard Intel MULTIBUS board and chassis products, comes complete with connections to five user terminals (expandable to nine), a state-of-the-art 35MB Winchester disk, and a 1MB diskette drive. In addition to the standard OEM XENIX multi-user operating system, SYSTEM 86/380X is also supplied with an efficient "C" compiler, editor, debugger and all other standard XENIX software. Substantial expansion capacity is provided for both MULTIBUS boards and an additional 8-inch standard peripheral, allowing large computing systems to be built quickly to take advantage of rapidly-changing OEM opportunities.



* XENIX is a trademark of Microsoft Corporation.

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, IRMX, iSBC, iSBX, iSXM, MULTICHANNEL, MULTIMODULE and ICS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

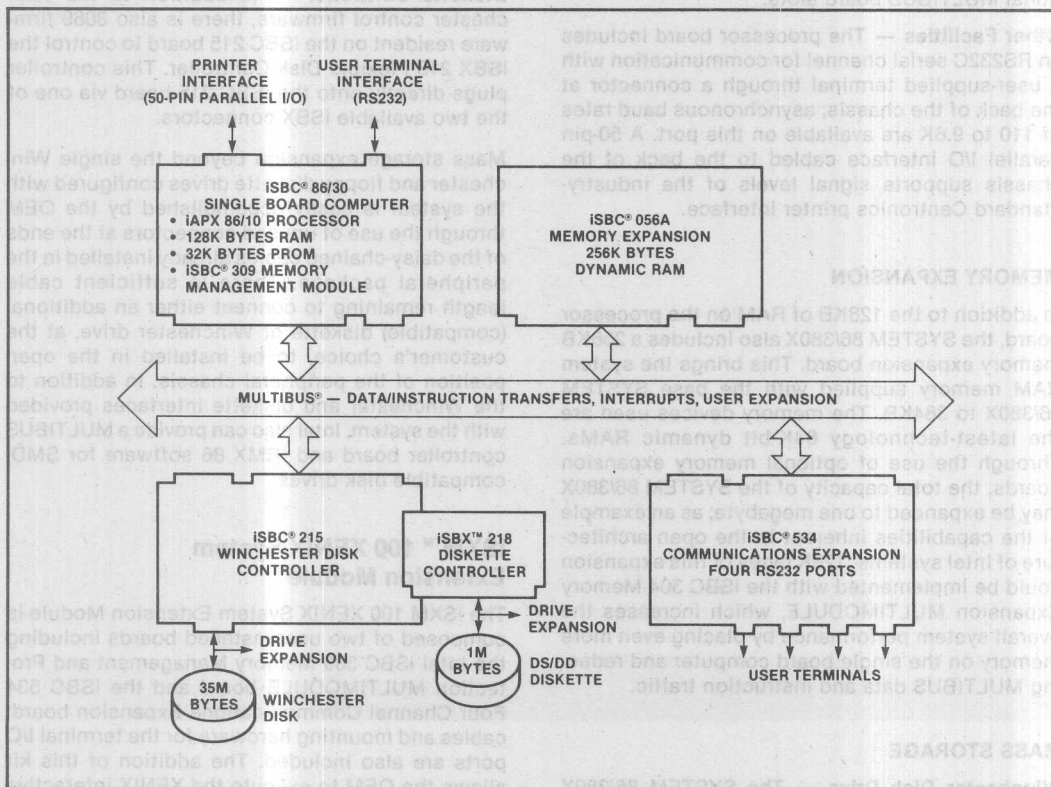
SYSTEM TECHNICAL DESCRIPTION

The SYSTEM 86/380X is a complete 16-bit microcomputer system designed to execute UNIX¹, the standard interactive multi-user operating system. The SYSTEM 86/380X operates under the control of XENIX, a fully-licensed implementation of Bell Laboratories UNIX V7 that has been designed specifically to execute on the high-performance IAPX 86/10 16-bit processor. The system's hardware is based on the Intel SYSTEM 86/380, plus the addition of the ISXM 100 XENIX System Extension Module, which includes four additional serial channels (expandable to nine), processor memory management and protection hardware, cables and mounting hardware. The addition of the Intel-supplied XENIX software and the Extension Module expand the capability of the system to execute this interactive, multi-user operating system.

¹UNIX is a trademark of Bell Laboratories.

Designed for OEM Open Systems Market Requirement

The SYSTEM 86/300 family of microcomputer systems are designed to meet the emerging OEM systems market requirement for "open" systems. Open-systems requirements demand that the system be (1) open to easily absorb the next two generations of VLSI microcomputers, (2) open to instantly leverage off industry standard hardware modules and interfaces (such as the MULTIBUS and ISBX buses), (3) open to easily tap industry standard software from Intel and independent vendors, (4) open to industry standard data communication interconnections, and (5) open to allow the OEM to participate at any level of integration: systems, boards and components. The open SYSTEM 86/300 microcomputer family provides the OEM with instant access to VLSI technology, the instant leverage and stability of software and hardware standards and documentation; and the only design approach to keep pace with VLSI technology.



SYSTEM 86/380X Block Diagram

Hardware

PROCESSOR SECTION

Central Processor — The controlling element of SYSTEM 86/380X is the iSBC 86/30 Single Board Computer. The central processor of the iSBC 86/30 Single Board Computer is the powerful 16-bit 8086, operating at 8 MHz. The architecture of this VLSI processor includes four 16-bit byte-addressable registers and two 16-bit index registers, accessible by a total of 24 operand addressing modes for complex data handling and flexible memory addressing.

Memory — Resident on the iSBC 86/30 board are 128KB of high-performance dual-ported RAM accessible to both the local and MULTIBUS system.

Expansion — The iSBC 86/30 board also supports one connector for the iSBX bus, an expansion bus allowing the user to add parallel or serial I/O, analog I/O, peripheral controllers, and other functions at low cost without requiring the use of additional MULTIBUS board slots.

Other Facilities — The processor board includes an RS232C serial channel for communication with a user-supplied terminal through a connector at the back of the chassis; asynchronous baud rates of 110 to 9.6K are available on this port. A 50-pin parallel I/O interface cabled to the back of the chassis supports signal levels of the industry-standard Centronics printer interface.

MEMORY EXPANSION

In addition to the 128KB of RAM on the processor board, the SYSTEM 86/380X also includes a 256KB memory expansion board. This brings the system RAM memory supplied with the base SYSTEM 86/380X to 384KB. The memory devices used are the latest-technology 64K-bit dynamic RAMs. Through the use of optional memory expansion boards, the total capacity of the SYSTEM 86/380X may be expanded to one megabyte; as an example of the capabilities inherent in the open architecture of Intel systems, 128K bytes of this expansion could be implemented with the iSBC 304 Memory Expansion MULTIMODULE, which increases the overall system performance by placing even more memory on the single board computer and reducing MULTIBUS data and instruction traffic.

MASS STORAGE

Winchester Disk Drive — The SYSTEM 86/380X contains a high performance 35MB (32MB for-

matted) 8-inch Winchester fixed-media disk drive for program and data storage. The drive has an average access time of 43 milliseconds and a transfer rate of 6.44 Mb/bs/sec.

Diskette Drive — An 8-inch double-density/double-sided diskette drive with 1M byte capacity is included in the base system. This high-density drive, which has an average access time of 91 milliseconds and a transfer rate of 500Kb/bs/sec, can be used for both data storage and system backup.

Intelligent Controller — The SYSTEM 86/380X uses the intelligent, 8089-based iSBC 215 Winchester Disk Controller. This high performance interface contains firmware which is executed directly on the 8089 I/O processor. This I/O processor, coupled with an onboard RAM buffer, offloads a significant portion of disk I/O overhead from the host 8086 processor.

Diskette Controller — In addition to the Winchester control firmware, there is also 8089 firmware resident on the iSBC 215 board to control the iSBX 218 Flexible Disk Controller. This controller plugs directly onto the iSBC 215 board via one of the two available iSBX connectors.

Mass storage expansion beyond the single Winchester and floppy diskette drives configured with the system is easily accomplished by the OEM through the use of unused connectors at the ends of the daisy-chained cables already installed in the peripheral package. There is sufficient cable length remaining to connect either an additional (compatible) diskette or Winchester drive, at the customer's choice, to be installed in the open position of the peripheral chassis. In addition to the Winchester and diskette interfaces provided with the system, Intel also can provide a MULTIBUS controller board and iRMX 86 software for SMD-compatible disk drives.

iSXM™ 100 XENIX System Extension Module

The iSXM 100 XENIX System Extension Module is composed of two user-installed boards including the Intel iSBC 309 Memory Management and Protection MULTIMODULE board and the iSBC 534 Four Channel Communications Expansion board; cables and mounting hardware for the terminal I/O ports are also included. The addition of this kit allows the OEM to execute the XENIX interactive operating system.

MEMORY MANAGEMENT AND PROTECTION

The Intel iSBC 309 board is designed to add memory management capability to the SYSTEM 86/380X microcomputers. The iSBC 309 board plugs directly into the processor socket of the iSBC 86/30 board, and it does not require any additional interface or power requirements.

The architecture of the iSBC 309 board is specially designed to optimize the performance of the XENIX operating system. There is a set of eight registers on the iSBC 309 board that make up a system process map cache. Through software support in the XENIX operating system, this cache contains the memory locations of not only the XENIX kernel, but also the seven most-recently-used user processes. The Intel implementation of XENIX constantly replaces the map of the least-used process with the map of the process about to become active. This cache of process map addresses allows the XENIX operating system to perform context switching between users and the operating system much faster than conventional memory management schemes that necessitate the swapping of user addresses through a single set of registers.

The iSBC 309 board provides two modes of 8086 operation: a system (or privileged) mode and a user mode. In the system mode the operating system can map the logical address space of a user mode process anywhere in the processor's one megabyte address space. The logical memory is mapped in 2K byte blocks. Each block can be marked as non-existent, read only, or read/write memory. These operations are performed automatically by the XENIX operating system and are transparent to the user.

In the user mode a process cannot alter the memory map or change the process number. Depending on the XENIX configuration, up to 7 user processes (plus one system process) with independent memory maps can be mapped at one time. The ability to support multiple process maps reduces the operating system overhead by reducing the time spent loading the memory map.

The XENIX operating system available from Intel takes full advantage of these powerful memory management features available on the iSBC 309 board. For more detailed information on the iSBC 309 board refer to the iSBC 308/309 Memory Management and Protection MULTIMODULE boards data sheet, number 210496-001.

Serial I/O Expansion

In addition to the one serial I/O channel resident on the iSBC 86/30 processor board, the iSXM 100 module provides four additional ports through the use of an Intel iSBC 534 Serial Expansion board. These ports are easily connected to any standard RS232C terminal. When in a local environment, asynchronous data rates up to 9.6K baud are supported.

The iSBC 534 board is shipped fully configured and ready to plug in the SYSTEM 86/380X system. The board interfaces directly to the system through the industry-standard (IEEE-796) MULTIBUS system bus. Additional iSBC 534 boards may be added by the user to expand beyond the 4 additional serial I/O ports supplied with the iSXM 100 package.

INSTALLATION

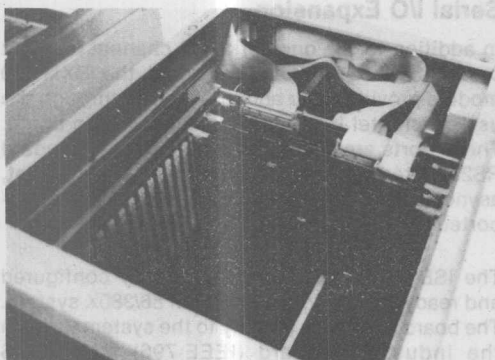
The iSXM 100 module is designed to be field installed into any iSBC 86/30-based SYSTEM 86/380 microcomputer. An iSXM 100 Installation Guide giving detailed instructions is included with the iSXM 100 Kit. Total installation time for the hardware kit is estimated to be less than one hour. Complete hardware installation can also be provided by Intel Field Service Engineers for an optional fee. Contact the local Intel Sales Office or Intel distributor for more information on field service installation.

System Chassis

SYSTEM PACKAGING CONFIGURATION

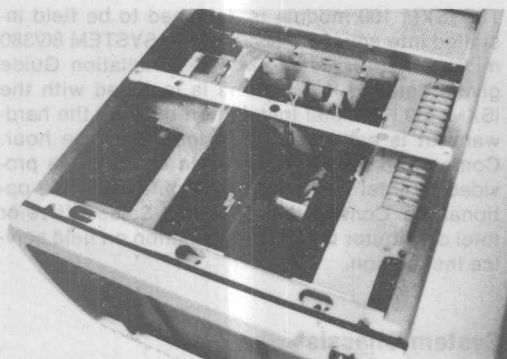
The SYSTEM 86/380 is packaged in two chassis.

Module Package — This package contains a 14-slot cardcage, all of whose slots accept MULTIBUS-compatible boards with iSBC MULTIMODULE expansion boards attached (See Figure 1). In addition, three of the slots are sufficiently far from adjacent slots to allow boards with iSBX MULTIMODULE boards to be inserted, with no penalty in loss of slots. Two of these wide slots and two standard slots are utilized by the boards which make up the SYSTEM 86/380X, leaving ten slots (one wide, nine narrow) for user expansion. I/O cables run from the boards in the Module Package to connectors on the back of the package; interconnecting cables run from that point to the back of the Peripheral Package.

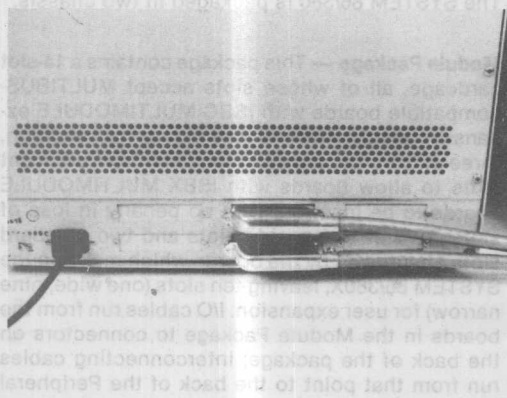


**Figure 1. Interior of SYSTEM 86/380X
Module Package**

Peripheral Package — This package houses the Winchester and diskette drives (see Figure 2). One position, suitable for mounting any industry-standard 8-inch peripheral drive, is available to the user.



**Figure 2. Interior of SYSTEM 86/380X
Peripheral Package**



The two chassis may be used on a desk top or mounted (with user-supplied slides) in a user-furnished 19-inch EIA equipment rack. The two packages are connected with a set of cables approximately 4.5 feet long. Knock-outs are available on the back panel of each chassis to allow the OEM to easily configure additional I/O connectors into the system (see Figure 3). The SYSTEM 86/380X has been designed to meet UL and CSA safety and FCC EMI/RFI requirements.

Software

GENERAL DESCRIPTION

The SYSTEM 86/380X executes the industry standard, interactive operating system, XENIX. XENIX is a fully-licensed Intel 8086 adaptation of the Bell Laboratories Version 7 UNIX operating system. The XENIX system is an interactive, protected multi-user, multitasking operating system with a powerful, flexible human interface. The operating system is well suited to applications requiring multiple persons running independent, terminal-oriented jobs. Example applications include distributed data processing, business data processing, software development and engineering or scientific data analysis.

The differences between XENIX and UNIX V7 can either be classified as improvements or enhancements over the standard Bell Laboratories product.

Improvements are changes that are made to enhance reliability and performance of the operating system but are transparent to the user. Enhancements are new features to the operating system which applications software programmers can take advantage of.

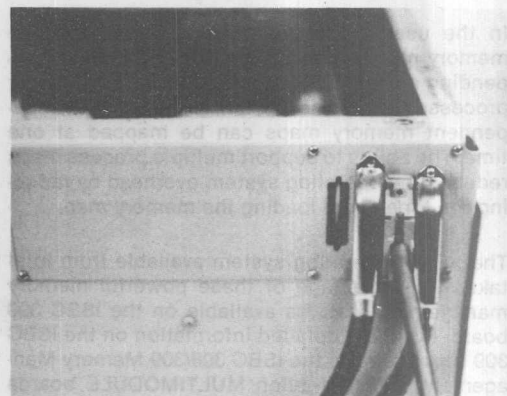


Figure 3. Cable Connector Panels — SYSTEM 86/380X Peripheral Package (left) and Module Package (right)

The major improvements and enhancements that are available in Intel's version of the XENIX operating system running on the SYSTEM 86/380X microcomputer are as follows:

Automatic Disk Recovery — XENIX writes information on the disk in a more structured manner. Unlike UNIX, XENIX is able to recover information on the disk in the event of a system crash (unless there is a media failure). A special program, FSCK, cleans and restructures an unhealthy disk.

Size and Speed — The implementation of XENIX used in the SYSTEM 86/380X has been optimized for both the 8086 microprocessor and for use on the hardware offered on the Intel system. Specific improvements have been made in such areas as memory management, serial I/O drivers and disk drivers to name a few.

Floating Point — Floating point routines are moved from each individual program to a single copy located in the kernel. This saves memory space and improves system performance.

System Configuration — To make it easier to generate a new XENIX system there is a new interactive configuration utility supplied with the system to allow the user to specify device drivers, disk buffers, memory size etc. For more information on other enhancements and improvements to the XENIX operating system please refer to the Intel XENIX data sheet.

Scatter Loading — In conjunction with the iSBC 309 Memory Management board used in the SYSTEM 86/380X a sophisticated memory allocation algorithm is used to maximize memory utilization and minimize system swapping. The eight register sets that make up the process map cache on the iSBC 309 are constantly replacing the least used process address with the address of a process that is currently swapped out and wants to execute. Unlike most minicomputer implementations of UNIX, which need large contiguous blocks of memory, XENIX running on the SYSTEM 86/380X can load multiple 2K-byte blocks of a user's program in non-contiguous memory. This allows the operating system to utilize memory more efficiently and significantly reduces memory fragmentation and system swapping for higher performance.

The XENIX operating system programs can be described as being in three major categories:

Kernel — The kernel manages data storage and schedules tasks and manages main memory and peripherals.

Shell — This program is the human interface that interprets the commands typed by a user. The shell calls in user programs from storage and executes them one at a time or in a series called a "pipe".

Utility Programs — These programs perform a number of system routines such as compiling, editing, file maintenance, etc.

XENIX OPERATING SYSTEM FEATURES

Device Independent I/O — Application software is written with device independent I/O. This allows the user to define the peripheral device at run time.

Tree-structured File Directory and Task Hierarchies — Users can access information on secondary storage by referring to file with its ASCII name. The names of files stored on a device are stored in special files called directories. As directories are themselves named files, the XENIX file system allows directories to contain the names of other directories. This structure is useful for isolating collections of file to an application, and for tailoring system data to the requirements of users and applications sharing storage devices.

Re-entrant/Shared Code — The XENIX system automatically separates code and data. This allows both systems programs such as editors and compilers as well as user code to be both shared and re-entrant. This method of memory allocation results in a more efficient memory utilization and higher performance system.

System Accounting and Security Access Protection — The XENIX system has a complete security system for both system and file access. All data concerning system usage is also available for user accounting programs and system tuning.

In addition to the operating system itself, the XENIX system includes all of the support software developed by Bell Laboratories that has been added to the UNIX system during the last decade. It includes:

- Microsoft implementation of the Ritchie and Kernighan C compiler
- Text editing and typesetting software
- Subroutine libraries
- Assembler and debugger
- System software development tools

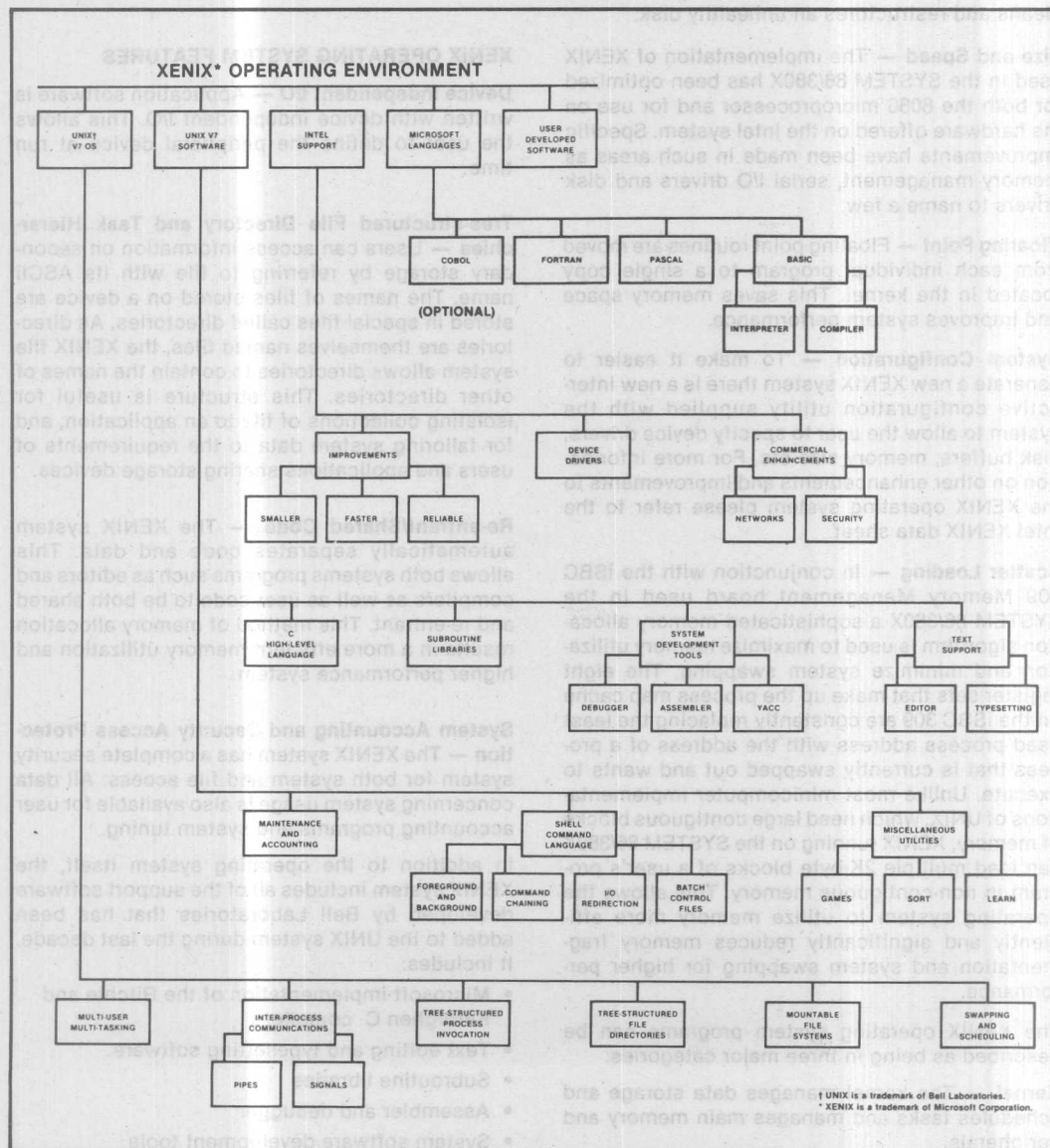
COMPREHENSIVE SELF-TEST AND SYSTEM DIAGNOSTICS

In order to ensure correct system operation and assist in rapid location of both hardware and software problems, the SYSTEM 86/380X is supplied with two levels of diagnostic routines:

SCT (System Confidence Test) — Automatically executed at power on and system reset, the SCT

performs a GO-NO GO condition for the processors, memory and devices in the system.

SDT (System Diagnostic Test) — In the event that the SCT finds a system problem, the SDT may be executed by the user for a detailed analysis of the hardware status. This easy-to-use monitor-based program can perform tests on all hardware boards and mass storage peripherals in the system.



SPECIFICATIONS

Word Size

GENERAL PURPOSE PROCESSOR

Instruction — 8, 16, or 32 bits

Data — 8/16 bits

Instruction Cycle Time

250 nanoseconds for fastest executable instructions (assuming 8 MHz clock rate and the instruction is in the queue).

750 nanoseconds for fastest executable instructions (assuming 8 MHz clock rate and the instruction is not in the queue).

Memory Capacity

RAM — 384K bytes supplied with the base system. Memory may be expanded to 1 megabyte.

Interface

EIA Standard RS232C signals provided and supported.

Serial — Five RS232C ports configurable from 110 to 9.6K baud asynchronous

Parallel — A 50-pin parallel I/O port that is supplied with signal levels compatible with the industry standard Centronics printer interface (OEM supplied cable)

AC Requirements

Module Package — 1738W maximum power consumption; 12.5A @ 92-126 VAC, 60 Hz single-phase (US); 6.3A @ 184-252 VAC, 50 Hz (Eur); 13.8 A @ 92-126 VAC, 50 Hz single-phase (Japan)

Peripheral Package — 693W maximum power consumption; 5.0A @ 92-126 VAC, 60 Hz single-phase (US); 2.5A @ 184-252 VAC, 50 Hz (Eur); 5.5A @ 92-126 VAC, 50 Hz single-phase (Japan)

Product Safety Standards

The system is designed to meet UL standard 114 Safety of Electronic Data Processing Units and Systems, plus the Canadian Standards Association standard C22.2 154-1975 Safety of Data Pro-

cessing Equipment. The system is also designed to meet the applicable RFI/EMI requirements of VDE 0871/6.78, VDE 0875/6.77 and FCC rule 47 CFR part 15 subpart J Emission Limits for Computing Devices.

Environmental Requirements

OPERATING

Temperature — 15°C to 35°C

Relative Humidity — 20% to 80% non-condensing over the operating temperature range*

Vibration — 0.01g inches peak-to-peak, 5 - 25 Hz; 0.2g 0-to-peak, 25 - 65 Hz; 0.3g 0-to-peak, 65 - 300 Hz

*NOTE: The environmental combination of humidity and temperature together cannot exceed 26°C wet bulb.

NON-OPERATING

Temperature — -25°C to 60°C

Relative Humidity — 20% to 80% non-condensing

Vibration — .020 inches, peak-to-peak, 5 - 25 Hz; .010 inches, peak-to-peak, 25 - 65 Hz; 2.0g, 0-to-peak, 65 - 300 Hz

Physical Characteristics

Both the Module Package and the Peripheral Package have the same dimensions.

Width — 16.8 in. (42.6 cm)

Height — 12.2 in. (31.1 cm)

Depth — 23.0 in. (58.4 cm)

Weight — Module Package - 55 lb (25 kg); Peripheral Package - 70 lb (32 kg)

In addition to these two Packages, the chassis interconnecting cables weigh approximately 5 lb (3 kg).

ORDERING INFORMATION

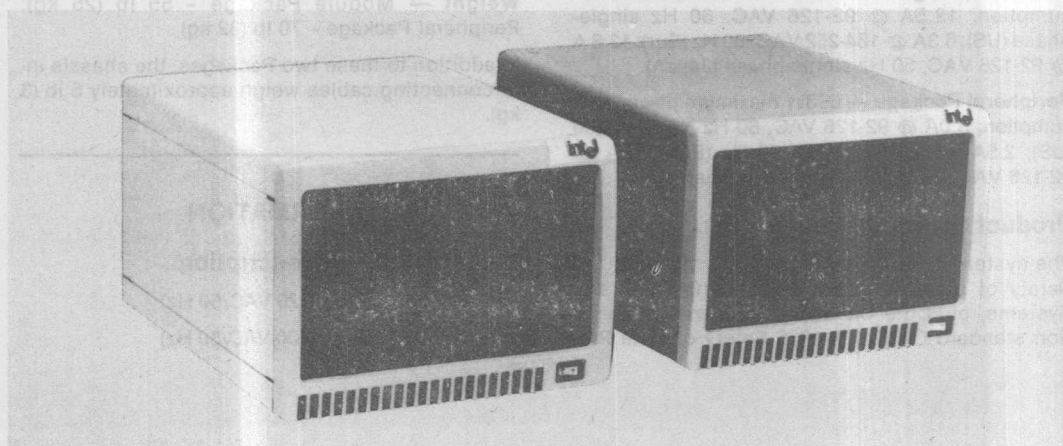
Part Number	Description
SYS86380ABX KIT	(220 VAC/50 Hz)
SYS86380ACX KIT	(100 VAC/50 Hz)



SYSTEM 86/380 MICROCOMPUTER SYSTEM iRMX™ 86 OPERATING SYSTEM

- Meets the open system requirements of the OEM
- Expansion provided for eleven iSBC® boards and one 8-inch standard peripheral in two desk-top or rack-mount chassis
- MULTIBUS® system bus multiprocessor architecture
- Open to VLSI extensions through standard hardware modules and interfaces
- High-performance computing with the iAPX 86/20 processor set (8086 16-bit CPU and 8087 numeric processor)
- Open to software extensions with Intel® (PASCAL-86, FORTRAN-86) and independent software vendor (BASIC, COBOL, C) languages
- Supported by iRMX™ 86, Intel's VLSI operating system
- 35MB Winchester disk drive and 1MB diskette drive for program and data storage, plus backup
- 384KB of high-speed RAM memory for multiple job and task execution
- Extensive self-test routines to ensure reliable operation and enhance fault isolation

The SYSTEM 86/380 Microcomputer System is a comprehensive, integrated hardware/software package designed for the OEM, yielding all the benefits of the latest advances in VLSI technology and performance without requiring complex system integration. Substantial expansion capacity is provided for both MULTIBUS boards and an additional 8-inch standard peripheral, allowing large computing systems to be built quickly to take advantage of rapidly-changing OEM opportunities. Based on industry standards established for the MULTIBUS system bus (IEEE-796) and 8-inch peripherals, the system also provides standardized software interfaces through the Universal Development Interface (UDI) available in Intel's iRMX 86 VLSI real-time multitasking operating system. Computing performance three to five times that of smaller minicomputers is made possible by the iAPX 86/20 VLSI processor set, supported by the large RAM memory system and high-capacity mass storage devices.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, iSXM, MULTICHANNEL, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

SYSTEM TECHNICAL DESCRIPTION

Designed for OEM Open Systems Market Requirement

The SYSTEM 86/300 family of microcomputer systems are designed to meet the emerging OEM systems market requirement for "open" systems. Open-systems requirements demand that the system be (1) open to easily absorb the next two generations of VLSI microcomputers, (2) open to instantly leverage off industry standard hardware modules and interfaces (such as the MULTIBUS and ISBX buses), (3) open to easily tap industry standard software from Intel and independent vendors, (4) open to industry standard data communication interconnections, and (5) open to allow the OEM to participate at any level of integration: systems, boards and components. The open SYSTEM 86/300 microcomputer family provides the OEM with instant access to VLSI technology, the instant leverage and stability of software and hardware standards and documentation and the

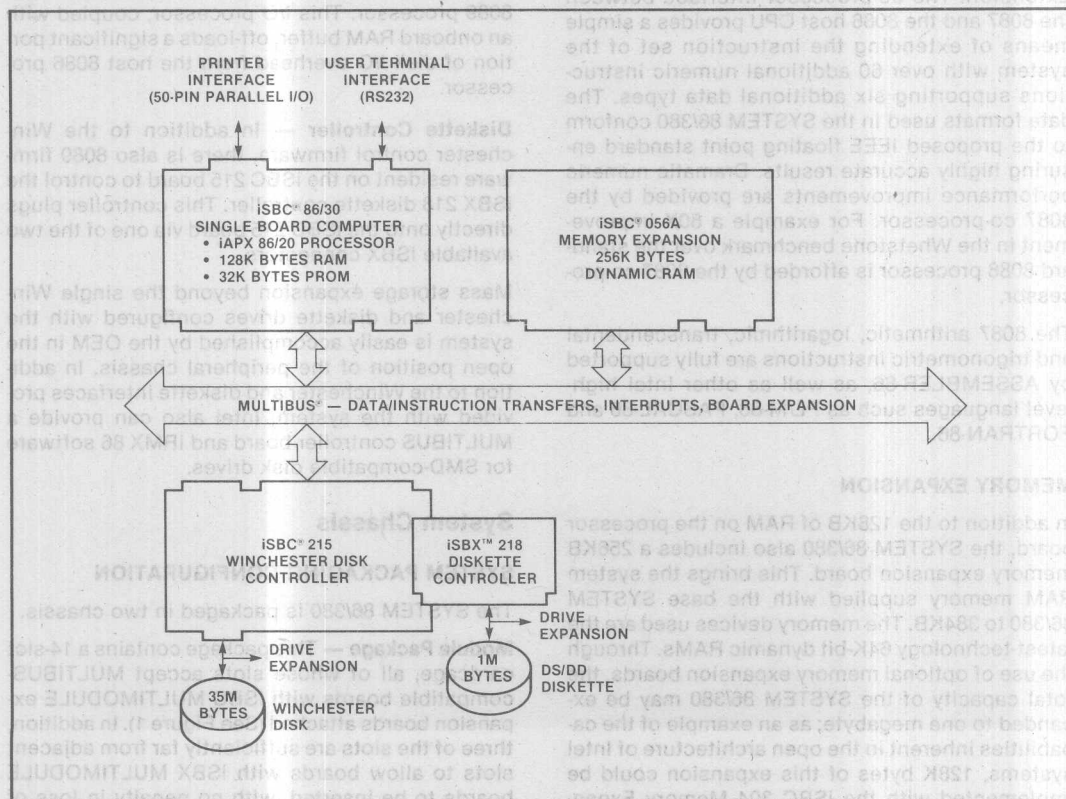
only design approach to keep pace with VLSI technology.

Hardware

PROCESSOR SECTION

Central Processor — The controlling element of SYSTEM 86/380 is the iSBC 86/30 Single Board Computer. The central processor of the iSBC 86/30 board is the powerful 16-bit 8086, operating at 5 MHz with the 8087 Numeric Processor Extension installed on the iSBC 337 Math expansion board. The architecture of this VLSI processor includes four 16-bit byte-addressable registers and two 16-bit index registers, accessible by a total of 24 operand addressing modes for complex data handling and flexible memory addressing.

Memory — Resident on the iSBC 86/30 board are 128KB of high-performance dual-ported RAM accessible to both the local and MULTIBUS system.



SYSTEM 86/380 Block Diagram

iSBX™ Expansion — The iSBC 86/30 board also supports two connectors for the iSBX bus, an expansion bus allowing the user to add parallel or serial I/O, analog I/O, peripheral controllers, and other functions at low cost and without requiring the use of additional MULTIBUS board slots.

Other Facilities — The processor board includes an RS232C serial channel for communication with a user-supplied terminal, through a connector at the back of the chassis. The SYSTEM 86/380 software configures this channel automatically for 9.6K baud, but asynchronous baud rates of 110 to 19.2K are available on the board. A 50-pin parallel I/O interface cabled to the back of the chassis supports signal levels of the industry-standard Centronics printer interface.

NUMERIC PROCESSING EXTENSION

Also included with the SYSTEM 86/380 is the iSBC 337 Numeric Data Processor. This MULTIMODULE board contains the Intel 8087 Numeric Processor Extension. The co-processor interface between the 8087 and the 8086 host CPU provides a simple means of extending the instruction set of the system with over 60 additional numeric instructions supporting six additional data types. The data formats used in the SYSTEM 86/380 conform to the proposed IEEE floating point standard ensuring highly accurate results. Dramatic numeric performance improvements are provided by the 8087 co-processor. For example a 50X improvement in the Whetstone benchmark over the standard 8086 processor is afforded by the 8087 co-processor.

The 8087 arithmetic, logarithmic, transcendental and trigonometric instructions are fully supported by ASSEMBLER-86, as well as other Intel high-level languages such as PL/M-86, PASCAL-86 and FORTRAN-86.

MEMORY EXPANSION

In addition to the 128KB of RAM on the processor board, the SYSTEM 86/380 also includes a 256KB memory expansion board. This brings the system RAM memory supplied with the base SYSTEM 86/380 to 384KB. The memory devices used are the latest-technology 64K-bit dynamic RAMs. Through the use of optional memory expansion boards, the total capacity of the SYSTEM 86/380 may be expanded to one megabyte; as an example of the capabilities inherent in the open architecture of Intel systems, 128K bytes of this expansion could be implemented with the iSBC 304 Memory Expansion MULTIMODULE, which increases the overall

system performance by placing even more memory on the single board computer and reducing MULTIBUS data and instruction traffic.

MASS STORAGE

Winchester Disk Drive — The SYSTEM 86/380 contains a high performance 35MB (32MB formatted) 8-inch Winchester fixed-media disk drive for program and data storage. The drive has an average access time of 43 milliseconds and a transfer rate of 6.44 Mbits/sec.

Diskette Drive — An 8-inch double-density/double-sided diskette drive with 1M byte capacity is included in the base system. This high-density drive, which has an average access time of 91 milliseconds and a transfer rate of 500Kbits/sec, can be used for both data storage and system backup.

Intelligent Controller — The SYSTEM 86/380 uses the intelligent, 8089-based iSBC 215 Winchester controller. This high performance interface contains firmware which is executed directly on the 8089 processor. This I/O processor, coupled with an onboard RAM buffer, off-loads a significant portion of disk I/O overhead from the host 8086 processor.

Diskette Controller — In addition to the Winchester control firmware, there is also 8089 firmware resident on the iSBC 215 board to control the iSBX 218 diskette controller. This controller plugs directly onto the iSBC 215 board via one of the two available iSBX connectors.

Mass storage expansion beyond the single Winchester and diskette drives configured with the system is easily accomplished by the OEM in the open position of the peripheral chassis. In addition to the Winchester and diskette interfaces provided with the system, Intel also can provide a MULTIBUS controller board and iRMX 86 software for SMD-compatible disk drives.

System Chassis

SYSTEM PACKAGING CONFIGURATION

The SYSTEM 86/380 is packaged in two chassis.

Module Package — This package contains a 14-slot cardcage, all of whose slots accept MULTIBUS-compatible boards with iSBC MULTIMODULE expansion boards attached (See Figure 1). In addition, three of the slots are sufficiently far from adjacent slots to allow boards with iSBX MULTIMODULE boards to be inserted, with no penalty in loss of slots. Two of these wide slots and a third standard

slot are utilized by the boards which make up the SYSTEM 86/380, leaving eleven slots (one wide, ten narrow) for user expansion. I/O cables run from the boards in the module package to connectors on the back of the package; interconnecting cables run from that point to the back of the peripheral package.

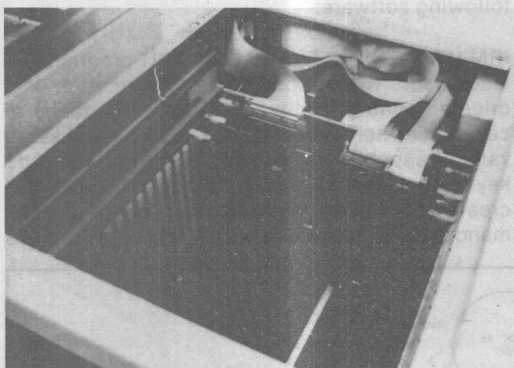


Figure 1. Interior of SYSTEM 86/380 Module Package

Peripheral Package — This package houses the Winchester and diskette drives (see Figure 2). One position, suitable for mounting any industry-standard 8-inch peripheral drive, is available to the user. The two chassis may be used on a desk top or mounted (with user-supplied slides) in a user-furnished 19" EIA equipment rack. The two packages are connected with a set of cables approximately 4.5 feet long. Knock-outs are available on the back panel of each chassis to allow the OEM to easily configure additional I/O connectors into the system (see Figure 3). The SYSTEM 86/380 has been designed to meet UL and CSA safety and FCC EMI/RFI requirements.

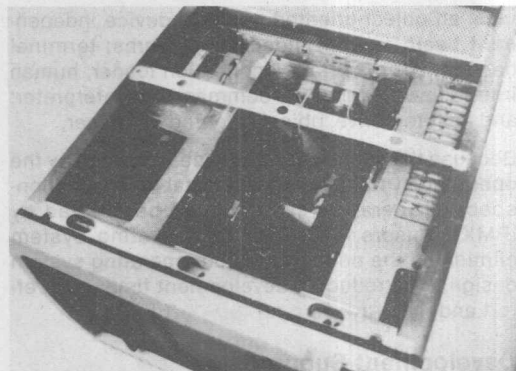
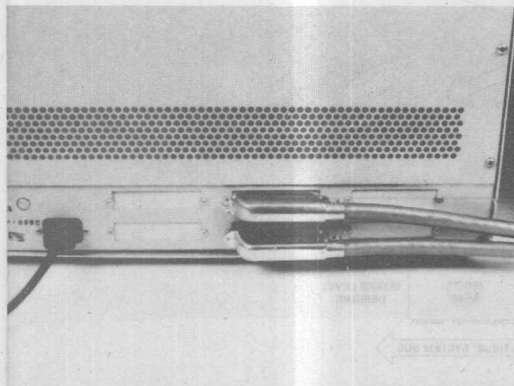


Figure 2. Interior of SYSTEM 86/380 Peripheral Package

System Software

VLSI OPERATING SYSTEM (iRMX™ 86)

The powerful iRMX 86 Operating System is an easy-to-use, comprehensive multiprogramming software system designed not only for the SYSTEM 86/380, but for iAPX 86/88-based iSBC board and component level designs.

Services provided by the RAM-based iRMX 86 Operating System include facilities for executing programs concurrently, sharing resources and information, servicing asynchronous events, and interactively controlling system resources and utilities. In addition, the iRMX 86 Operating System provides all major real-time facilities including priority-based system resource allocation, means for concurrently monitoring and controlling multiple external events, real-time clock control, interrupt management, and task dispatching. The iRMX 86 Operating System contains the following mod-

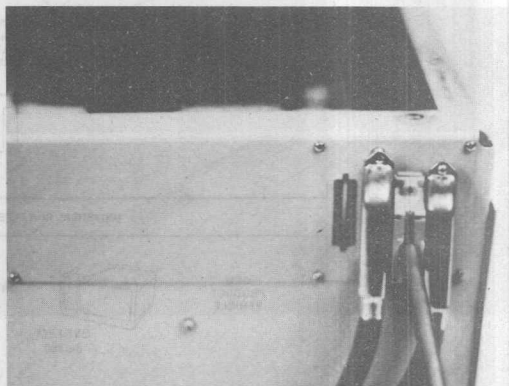


Figure 3. Cable Connector Panels — SYSTEM 86/380 Peripheral Package (left) and Module Package (right)

ules: an object-oriented nucleus; device independent basic and extended I/O systems; terminal handler; bootstrap and application loader; human interface with complete command line interpreter; and an interactive, object-oriented debugger.

Because the modules and services provided by the operating system are user-selectable, application-specific operating systems can be created by iRMX 86 users. The iRMX 86 Operating System eliminates the need for custom operating system design, thus reducing development time, cost, effort and risk.

Development Support

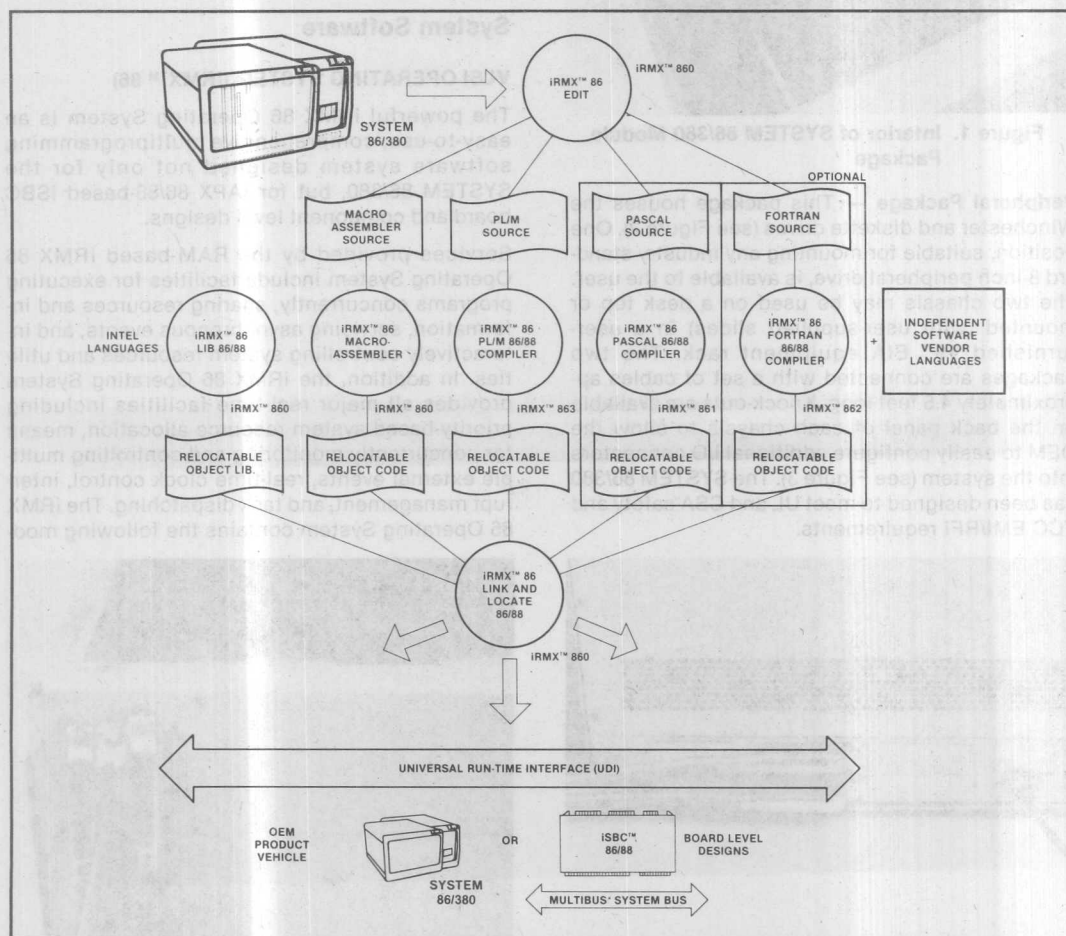
In addition to its capabilities as an OEM operating system, SYSTEM 86/380 and the iRMX 86 Operat-

ing System provide application development languages and facilities to the OEM to allow development on the target system.

iRMX™ UTILITIES PACKAGE (iRMX™ 860)

The iRMX 860 Utilities Package consists of the following software:

iRMX™ 86 EDIT — The iRMX 86 EDIT program provides users with a powerful, sophisticated, line-oriented editing facility. EDIT delivers a range of capabilities suitable for novice users as well as advanced capabilities for sophisticated users. Its key features include a macro processor capable of creating and executing complex strings of commands, which eases the editing chore, as well as



SYSTEM 86/380 Microcomputer System

defining blocks of text which may be included anywhere in the text file. EDIT offers variable command sourcing, symbolic line numbering and reference by symbol. The facilities of EDIT allow users to create, maintain and manipulate extensive libraries of source code with minimal effort.

IRMX™ 86 LINK/LOCATE — The iRMX 86 LINK/LOCATE program connects object modules which have been individually compiled into a single, relocatable object module. The input object code may have been produced by any object module format-compatible compiler. Output object modules may be recombined into larger object modules, allowing work from a large programming staff to be easily integrated into an application system.

IRMX™ 86 LIB — the iRMX 86 LIB "Library Manager" allows creation and maintenance of object module libraries. These libraries allow easy collection of related object code to reduce the overhead of maintaining many separate modules. Users may create new libraries, add and delete object modules, as well as list the contents of the library and their public symbols.

PL/M-86 (iRMX™ 863)

The PL/M-86 compiler provides users with a powerful, microcomputer-oriented system programming language. The PL/M-80 language was introduced in 1976 by Intel. It was the first microcomputer-oriented, block structured, high-level language available. Since 1976, thousands of users, shipping over millions of microcomputer-based systems have generated their system software with PL/M-80 and PL/M-86.

PL/M-86 is a compatible superset of PL/M-80 which offers easy portability of software across the full range of microcomputers supplied by Intel. For more information about PL/M-86, see the PL/M 86/88 Software Package data sheet (402175).

FULL REALMATH SUPPORT

The iRMX 86 languages support the REALMATH floating point standard. This allows users of all iRMX 86 languages to access the iAPX 86/20 Numeric Data Processor using the iSBC 337 MULTIMODULE board. This numeric processor offers over 100 times greater performance than comparable software-implemented algorithms, and reduces the system memory requirements by at least 16KB. The REALMATH standard (proposed IEEE standard) provides universal consistency in

results of numeric computations. The iRMX 86 languages provide efficient object code generation and access to the highest-performance floating point package available on microcomputers.

COMPREHENSIVE SELF-TEST AND SYSTEM DIAGNOSTICS

In order to ensure correct system operation and rapid location of both hardware and software problems, the SYSTEM 86/380 is supplied with three levels of diagnostic routines:

SCT (System Confidence Test) — Automatically executed at power on and system reset, the SCT performs a GO-NO GO check for the processors, memory and peripherals in the system.

SDT (System Diagnostic Test) — In the event that the SCT finds a system problem, the SDT may be executed by the user for a detailed analysis of the hardware status. This easy-to-use monitor-based program can perform tests on all hardware boards in the system and mass storage peripherals.

SAT (System Analysis Test) — This diagnostic tool tests the integration of the hardware and software at the system level. This helps to isolate intermittent failures by running a load stress test on both hardware and software.

SYSTEM DEBUGGING TOOLS

The iRMX 86 Operating System provides a comprehensive tool for interactive software debugging. The debugger has two capabilities that greatly simplify the process of debugging a multitasking system. First, the debugger allows users to debug several tasks while the balance of the application system continues to run in real-time. Second, the debugger allows programmers to interactively view and modify system constructs as well as the system RAM and CPU registers. The debugger is structured to enable system designers to track system-wide problems easily. It can also remain in the final OEM application as a maintenance tool.

OEM SOFTWARE LICENSING

The Intel software products listed above require the signing of an Intel Master Software License Agreement. All products include one year of update service. Software is shipped on floppy media in two object forms: 1) a ready-to-run, fully configured system and 2) a configurable version of all software products.

OPTIONAL INTEL® SOFTWARE SUPPORT

In addition to ASM-86 and PL/M-86 compilers included in the base SYSTEM 86/380 the following Intel languages are available for use on the system:

PASCAL-86 (iRMX™ 861) — The PASCAL-86 compiler provides a strict implementation of the proposed ISO language standard. All source programs are validated by the compiler to ensure its conformance to the standard. Many extensions to the language are available which allow PASCAL programs to be written specifically for microcomputers. Separate module compilation and iAPX 86/20, 88/20 Numeric Data Processor support are a few of its many features. The ISO standard "source evaluator" can be switched off to accept these extensions. For more information on iRMX 86 PASCAL features, see the PASCAL 86/88 Software Package data sheet (121680).

FORTRAN-86 (iRMX™ 862) — The iRMX 86 FORTRAN compiler provides users total compatibility with existing FORTRAN 86 language-generated code, plus many new language features provided by the FORTRAN 77 language standard. These new features offer FORTRAN programmers many new capabilities, including "IF-THEN-ELSE", random access I/O and character variables. For a more detailed explanation of iRMX 86 FORTRAN, see the FORTRAN 86/88 Software Package data sheet (400630).

MULTIPROCESSING EXPANSION

iMMX 800 MULTIBUS® Message Exchange — The advanced design of the MULTIBUS system bus coupled with the iMMX 800 software package allows the SYSTEM 86/380 to easily support additional 8 and/or 16-bit Intel single board computers. This powerful option enables OEMs to increase the SYSTEM 86/380's general purpose processing power with boards such as the iSBC 86/05 (8 MHz 8086) and iSBC 88/25 (5 MHz 8088), or the iSBC

* ETHERNET is a trademark of Xerox Corporation.

SPECIFICATIONS

Word Size

GENERAL PURPOSE PROCESSOR

Instruction — 8, 16, or 32 bits

Data — 8/16 bits

NUMERIC CO-PROCESSOR

Data — 80 bits

80/24 (5 MHz 8085). Intelligent, high-performance microprocessor-based boards such as the Ethernet® communications controller (iSBC 550 board), serial communications controller (iSBC 544 board) and analog measurement and control computer (iSBC 88/40 board) can greatly expand the capabilities and processing power of the SYSTEM 86/380.

Independent Software Vendor Support — Through the use of the standard UDI (Universal Development Interface) a wide variety of language products, both compilers and interpreters, are now available for use on iRMX 86 from independent software vendors. COBOL, BASIC and C are a few examples of the available languages. Contact your Intel sales representative for more information.

System Options

HARDWARE

Any Intel MULTIBUS board may be installed by the OEM into the eleven available expansion slots in the system chassis. Over 40 MULTIBUS boards are available from Intel, and more than 90 other vendors produce MULTIBUS-compatible products. In addition, Intel manufactures iSBX Bus MULTIMODULE boards for use in any board's iSBX Bus connector, and iSBC Extension MULTIMODULE boards for memory and math co-processor additions which are tightly-coupled to each individual board's architecture.

SOFTWARE

In addition to the iRMX 86 operating system, the XENIX interactive multi-user operating system is also available for SYSTEM 86/380 computers. See the SYSTEM 86/380X data sheet. Optional language products are available from both Intel and independent software vendors. Operating system support for additional 8085-, 8088- and 8086-based processor boards is available through the use of the iRMX 80 and iRMX 88 Operating Systems.

Instruction Cycle Time

400 nanoseconds for fastest executable instructions (assuming 5 MHz clock rate and the instruction is in the queue).

1.0 microsecond for fastest executable instructions (assuming 5 MHz clock rate and the instruction is not in the queue).

Memory Capacity

RAM — 384K bytes supplied with the base system. Memory may be expanded to 1 megabyte.

Interface

EIA Standard RS232C signals provided and supported.

Serial — Configurable from 110 to 19.2K baud asynchronous; SYSTEM 86/380 software configures the user terminal port for 9.6K baud

Parallel — A 50-pin parallel I/O port that is supplied with signal levels compatible with the industry standard Centronics printer interface (OEM supplied cable)

AC Requirements

Module Package — 1738W maximum power consumption; 10A @ 92-126 VAC, 60 Hz, single-phase (US); 6.3A @ 184 to 252 VAC, 50 Hz, (Eur); 13.8A @ 92-126 VAC, 50 Hz single-phase (Japan)

Peripheral Package — 693W maximum power consumption; 7.0A @ 92-126 VAC, 60 Hz single-phase (US); 2.5A @ 184-252 VAC, 50 Hz (Eur); 7.0A @ 92-126 VAC, 50 Hz single-phase (Japan)

Product Safety Standards

The system is listed under UL standard 114 Safety of Electronic Data Processing Units and Systems. It is also certified by the Canadian Standards Association, standard C22.2 154-1975 Safety of Data Processing Equipment. The system complies with the International Electronics Commission standard IEC 435 Safety of Data Processing Equipment. The system also conforms to the applicable RFI/EMI requirements of FCC rule 47 CFR part 15 subpart J, Emission Limits for Computing Devices.

Environmental Requirements

OPERATING

Temperature and Relative Humidity — 15°C to 35°C dry bulb and 20% relative humidity to 80% relative humidity non-condensing subject to a maximum wet bulb temperature of 26°C. These conditions translate to a maximum relative humidity of 80% non-condensing

up to 28.8°C which falls to a maximum relative humidity of 50% at 35°C

Altitude — Sea level to 10,000 feet

Vibration — 0.0014 inches peak-to-peak, 5-25 Hz; 0.0007 inches peak-to-peak, 25-55 Hz; 0.3g 0-to-peak, 55-300 Hz

Shock — 1.0g for 11 ms

NON-OPERATING

Temperature — 25°C to 60°C

Relative Humidity — 20% to 80% non-condensing

Altitude — Sea level to 40,000 feet

Vibration — 0.008 inches peak-to-peak, 5-25 Hz; 0.004 inches peak-to-peak, 25-55 Hz; 2.0g 0-to-peak, 55-300 Hz

Shock — 15g for 11 ms

Physical Characteristics

Both the module package and the peripheral package have the same dimensions:

Width — 16.8 in. (42.6 cm)

Height — 12.2 in. (31.1 cm)

Depth — 21.0 in. (53.3 cm)

Weight — Module Package - 55 lb (25 kg); Peripheral Package - 70 lb (32 kg)

In addition to these two Packages, the chassis interconnecting cables weigh approximately 5 lb (3 kg).

Extensive System Documentation

The SYSTEM 86/380 is shipped with six documentation binders containing over 28 in-depth manuals on all aspects of hardware and software operation. A System Installation and Maintenance manual, in addition to a System Overview manual, is also included.

ORDERING INFORMATION

Part Number Description

SYS86380ABR KIT (220 VAC/50 Hz)

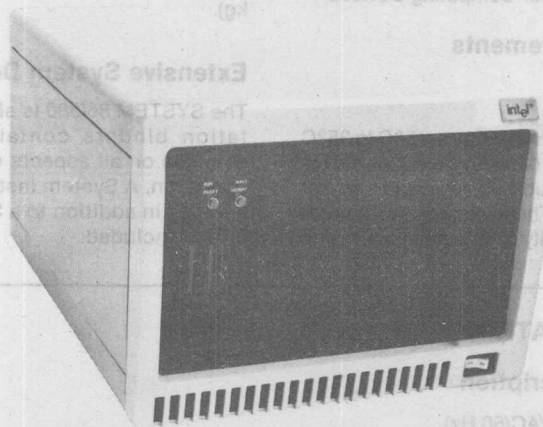
SYS86380ACR KIT (100 VAC/50 Hz)



SYSTEM 86/330X MICROCOMPUTER SYSTEM XENIX*

- Meets the open system requirements of the OEM
- Industry standard XENIX interactive operating system
- Memory management and protection for the high performance 8 MHz 16-bit iAPX 86/10 processor
- Standard language support includes C and ASSEMBLER-86. Other languages available soon
- MULTIBUS® system bus (IEEE-796) multiprocessor architecture
- 35MB Winchester and 1MB DS/DD 8" floppy for program, data storage and back-up
- 384KB of high speed RAM memory to execute multiple-user applications
- Extensive self-test routines for reliable operation and simple fault isolation
- 5 serial ports included (expandable to nine ports)

The Intel SYSTEM 86/330X Microcomputer System is an integrated, high-performance 16-bit XENIX-based microcomputer system. The system, which is based on standard Intel MULTIBUS system bus board level products, comes complete with connections to five user terminals (expandable to nine), a state-of-the-art 35MB Winchester disk, and a 1MB DS/DD floppy disk subsystem. In addition to the standard OEM multi-user operating system the SYSTEM 86/330X is also supplied with an efficient "C" compiler, editor, debugger and all other standard XENIX software.



* XENIX is a trademark of Microsoft Corporation.

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, iSXM, MULTICHANNEL, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

INTEL CORPORATION, 1982

SYSTEM TECHNICAL DESCRIPTION

Designed for OEM Open Systems Market Requirement

The SYSTEM 86/300 family of microcomputer systems are designed to meet the emerging OEM systems market requirement for "open" systems. Open-systems requirements demand that the system be (1) open to easily absorb the next two generations of VLSI microcomputers, (2) open to instantly leverage off industry standard hardware modules and interfaces (such as MULTIBUS and ISBX buses), (3) open to easily tap industry standard software from Intel and independent vendors, (4) open to industry standard data communication interconnections, and (5) open to allow the OEM to participate at any level of integration, systems, boards and components. The open SYSTEM 86/300 microcomputer family provides the OEM with instant access to VLSI technology, and the instant leverage and stability of software and hardware standards and documentation.

Open Systems Architecture Allows Multiple Levels of Integration

The entire SYSTEM 86/330X, both hardware and software, is built using standard, modular off-the-shelf Intel board level products and system soft-

ware. The system is designed so that the OEM can easily tailor a custom system through the use of individual Intel board level products.

The SYSTEM 86/330X is a complete 16-bit microcomputer system designed to execute UNIX[†], the 16-bit standard interactive multi-user operating system. The SYSTEM 86/330X uses XENIX, a fully-licensed implementation of Bell Laboratories UNIX V7 that has been designed specifically to execute on the high-performance iAPX 86/10 16-bit processor. The hardware of this XENIX system is based on the Intel SYSTEM 86/330A. The addition of the Intel-supplied XENIX software and the iSXM 100 XENIX System Extension Module which includes four additional serial channels (expandable to nine), processor memory management and protection, internal cables and mounting hardware, expand the capability of the system to execute this interactive, multi-user operating system.

[†] UNIX is a trademark of Bell Laboratories.

Hardware

PROCESSOR

The single board computer used in the SYSTEM 86/330X is the MULTIBUS-based iSBC 86/30 board. The central processor of the iSBC 86/30 board is

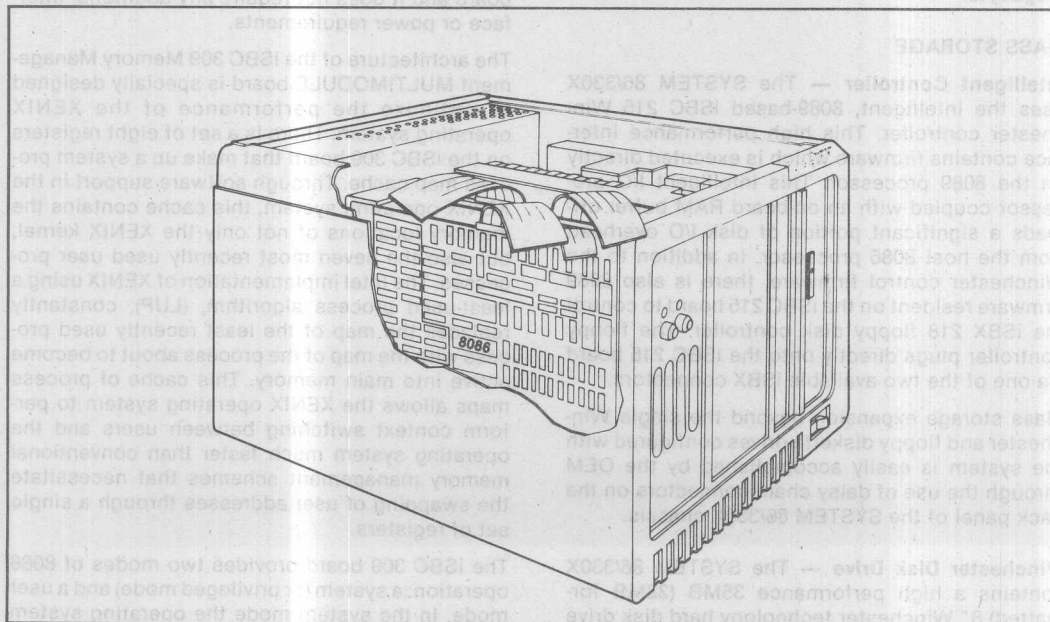


Figure 1. SYSTEM 86/330X Microcomputer System

the powerful 8 MHz 16-bit 8086. The architecture includes four 16-bit byte addressable registers, two 16-bit index registers, all accessed by a total of 24 operand addressing modes for complex data handling and flexible memory addressing.

The base SYSTEM 86/330X is supplied with 384KB of high speed RAM memory. There is 128KB of dual-ported RAM resident on the iSBC 86/30 processor board accessible to both the local and MULTIBUS system bus and can address up to one megabyte of main memory. The processor board includes an RS232C serial channel for the connection of a user supplied terminal. Cabling to the rear panel of the system is provided. Asynchronous baud rates of 110 to 9.6K are supported by the system. A parallel I/O interface is also available on the processor board.

MEMORY

In addition to the 128KB of dual-ported RAM on the iSBC 86/30 processor board, the SYSTEM 86/330X also includes a 256KB memory expansion board. This brings the system RAM memory supplied with the base SYSTEM 86/330X to 384KB. The memory used is the latest technology 64K bit dynamic RAMs. Through the use of optional memory expansion boards, the total memory capacity of the SYSTEM 86/330X may be expanded to one megabyte.

MASS STORAGE

Intelligent Controller — The SYSTEM 86/330X uses the intelligent, 8089-based iSBC 215 Winchester controller. This high performance interface contains firmware which is executed directly on the 8089 processor. This intelligent I/O processor coupled with an on-board RAM buffer off-loads a significant portion of disk I/O overhead from the host 8086 processor. In addition to the Winchester control firmware, there is also 8089 firmware resident on the iSBC 215 board to control the iSBX 218 floppy disk controller. The floppy controller plugs directly onto the iSBC 215 board via one of the two available iSBX connectors.

Mass storage expansion beyond the single Winchester and floppy diskette drives configured with the system is easily accomplished by the OEM through the use of daisy chain connectors on the back panel of the SYSTEM 86/330X chassis.

Winchester Disk Drive — The SYSTEM 86/330X contains a high performance 35MB (32MB formatted) 8" Winchester technology hard disk drive for program and data storage. The drive has an

average access time of 43 ms and a transfer rate of 6.44 Mbits/sec.

Floppy Diskette Drive — A double-density/double-sided, 8", 1MB, floppy disk drive is included in the base system. This high-density floppy drive, which has an average access time and a transfer rate of 500Kbits/sec, can be used for both data storage and system backup.

iSXM™ 100 XENIX System Extension Module

The iSXM 100 XENIX System Extension Module is composed of two user installed standard boards including the Intel iSBC 309 Memory Management MULTIMODULE board, iSBC 534 4-port Serial Expansion board, cables and mounting hardware. The addition of this kit allows the OEM to execute the XENIX interactive operating system.

MEMORY MANAGEMENT AND PROTECTION

The Intel iSBC 309 Memory Management MULTIMODULE board is designed to add memory management capability to the SYSTEM 86/330X microcomputers. The iSBC 309 board, which operates at either 5 or 8 MHz, plugs directly into the processor socket of the iSBC 86/30 processor board and it does not require any additional interface or power requirements.

The architecture of the iSBC 309 Memory Management MULTIMODULE board is specially designed to optimize the performance of the XENIX operating system. There is a set of eight registers on the iSBC 309 board that make up a system process map cache. Through software support in the XENIX operating system, this cache contains the memory locations of not only the XENIX kernel, but also the seven most recently used user processes. The Intel implementation of XENIX using a least-used process algorithm, (LUP), constantly replaces the map of the least recently used process with the map of the process about to become active into main memory. This cache of process maps allows the XENIX operating system to perform context switching between users and the operating system much faster than conventional memory management schemes that necessitate the swapping of user addresses through a single set of registers.

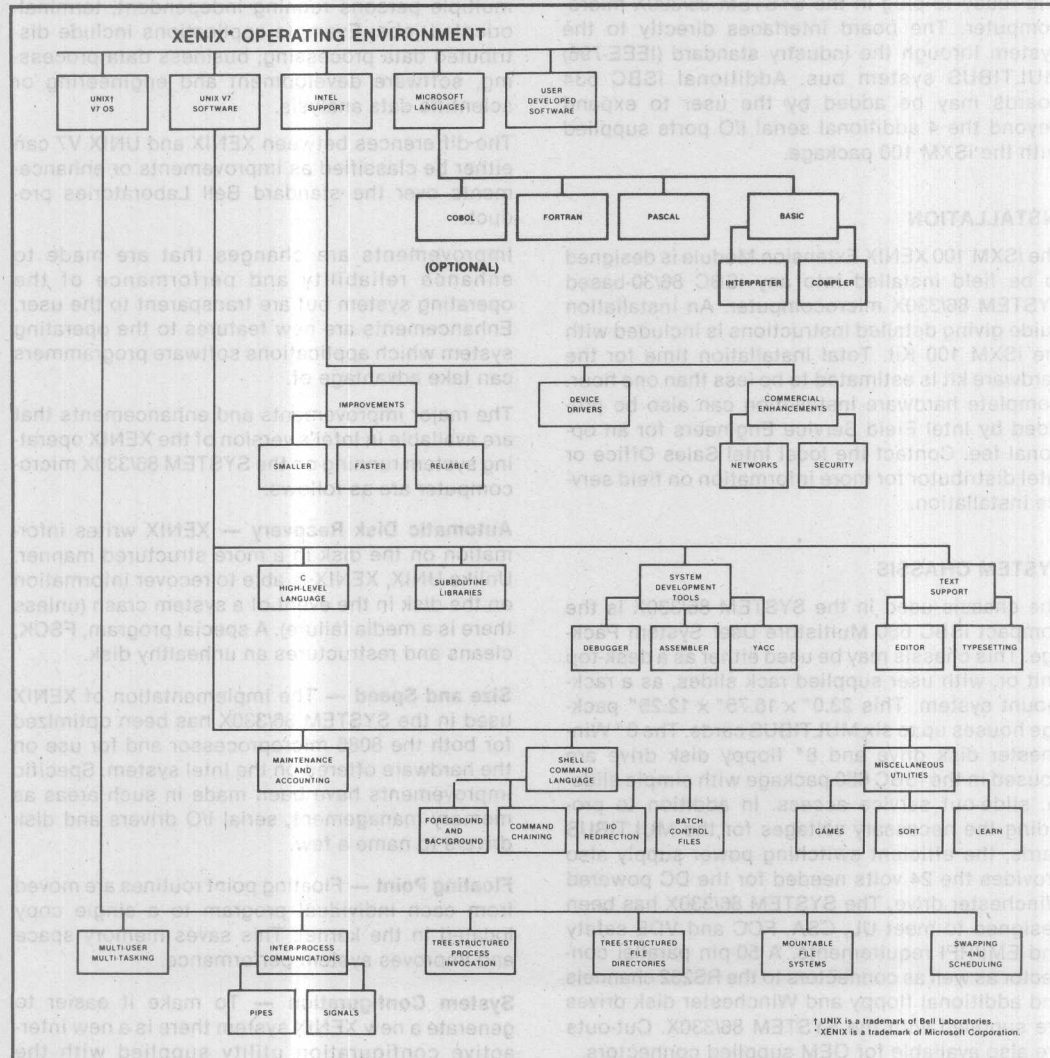
The iSBC 309 board provides two modes of 8086 operation: a system (or privileged mode) and a user mode. In the system mode the operating system can map the logical address space of a user mode

process anywhere in the processors one megabyte address space. The logical memory is mapped into 2K byte blocks. Each block can be marked as non-existent, read only, or read/write memory. These operations are performed automatically by the XENIX operating system and are transparent to the user.

In the user mode a process cannot alter the memory map or change the process number. Depending on the XENIX configuration, up to 7 user processes (plus one system process) with inde-

pendent memory maps can be actively mapped at one time. The ability to support multiple process maps reduces the operating system overhead by reducing the time spent loading the memory map.

The XENIX operating system available from Intel takes full advantage of these powerful memory management features available on the iSBC 309 board. For more detailed information on the iSBC 309 Memory Management board refer to the iSBC 308/309 Memory Management boards data sheet, number 210496-001.



Serial I/O

In addition to the one serial I/O channel resident on the iSBC 86/30 processor board, the iSXM 100 package provides four additional ports through the use of an Intel iSBC 534 Serial Expansion board. These ports are easily connected to any standard RS232C terminal. When in a local environment, asynchronous data rates up to 9600 baud are supported.

The iSBC 534 board is shipped fully configured and ready to plug in the SYSTEM 86/300X microcomputer. The board interfaces directly to the system through the industry standard (IEEE-796) MULTIBUS system bus. Additional iSBC 534 boards may be added by the user to expand beyond the 4 additional serial I/O ports supplied with the iSXM 100 package.

INSTALLATION

The iSXM 100 XENIX Extension Module is designed to be field installed into any iSBC 86/30-based SYSTEM 86/330X microcomputer. An installation guide giving detailed instructions is included with the iSXM 100 Kit. Total installation time for the hardware kit is estimated to be less than one hour. Complete hardware installation can also be provided by Intel Field Service Engineers for an optional fee. Contact the local Intel Sales Office or Intel distributor for more information on field service installation.

SYSTEM CHASSIS

The chassis used in the SYSTEM 86/330X is the compact iSBC 680 Multistore User System Package. This chassis may be used either as a desk-top unit or, with user supplied rack slides, as a rack-mount system. This 23.0" x 16.75" x 12.25" package houses up to six MULTIBUS cards. The 8" Winchester disk drive and 8" floppy disk drive are housed in the iSBC 680 package with simple slide-in, slide-out service access. In addition to providing the necessary voltages for the MULTIBUS cards, the efficient switching power supply also provides the 24 volts needed for the DC powered Winchester drive. The SYSTEM 86/330X has been designed to meet UL, CSA, FCC and VDE safety and EMI/RFI requirements. A 50-pin parallel connector as well as connectors to the RS232 channels and additional floppy and Winchester disk drives are supplied with the SYSTEM 86/330X. Cut-outs are also available for OEM supplied connectors.

Software

GENERAL DESCRIPTION

The SYSTEM 86/330X executes the industry standard, interactive operating system, XENIX. XENIX is a fully-licensed Intel 8086 adaptation of the Bell Laboratories Version 7 UNIX operating system. The XENIX system is an interactive, protected multi-user, multitasking operating system with a powerful, flexible human interface. The operating system is well suited to applications requiring multiple persons running independent, terminal-oriented jobs. Example applications include distributed data processing, business data processing, software development and engineering or scientific data analysis.

The differences between XENIX and UNIX V7 can either be classified as improvements or enhancements over the standard Bell Laboratories product.

Improvements are changes that are made to enhance reliability and performance of the operating system but are transparent to the user. Enhancements are new features to the operating system which applications software programmers can take advantage of.

The major improvements and enhancements that are available in Intel's version of the XENIX operating system running on the SYSTEM 86/330X microcomputer are as follows:

Automatic Disk Recovery — XENIX writes information on the disk in a more structured manner. Unlike UNIX, XENIX is able to recover information on the disk in the event of a system crash (unless there is a media failure). A special program, FSCK, cleans and restructures an unhealthy disk.

Size and Speed — The implementation of XENIX used in the SYSTEM 86/330X has been optimized for both the 8086 microprocessor and for use on the hardware offered on the Intel system. Specific improvements have been made in such areas as memory management, serial I/O drivers and disk drivers to name a few.

Floating Point — Floating point routines are moved from each individual program to a single copy located in the kernel. This saves memory space and improves system performance.

System Configuration — To make it easier to generate a new XENIX system there is a new interactive configuration utility supplied with the

system to allow the user to specify device drivers, disk buffers, memory size etc. For more information on other enhancements and improvements to the XENIX operating system please refer to the Intel XENIX data sheet number 210503-001.

Scatter Loading — In conjunction with the iSBX 309 Memory Management board used in the SYSTEM 86/330X a sophisticated memory allocation algorithm is used to ensure the maximum memory utilization and minimize system swapping. Unlike most minicomputer implementations of UNIX, which need large contiguous blocks of memory, XENIX running on the SYSTEM 86/330X can load multiple small 2K blocks of a users program into non-contiguous memory. This allows the operating system to utilize memory more efficiently and significantly reduces memory fragmentation and system swapping for higher performance.

The XENIX operating system programs can be described as being in three major categories:

Kernel — The kernel manages data storage, schedules tasks, manages main memory and peripherals.

Shell — This program is the human interface that interprets the commands typed by a user. The shell calls in user programs from storage and executes them one at a time or concurrently in a series called a pipe.

Utility Programs — These programs perform a number of system routines such as compiling, editing, file maintenance, etc.

XENIX OPERATING SYSTEM FEATURES

Device Independent I/O — Application software is written with device independent I/O. This allows the user to define the peripheral device at run time.

Tree-structured File Directory and Task Hierarchies — Users can access information on secondary storage by referring to a file with its ASCII name. The names of files stored on a device are stored in special files called directories. As directories are themselves named files, the XENIX file system allows directories to contain the names of other directories. This structure is useful for isolating collections of files particular to an application, and for tailoring system data to the requirements of users and applications sharing storage devices.

Re-entrant/Shared — The XENIX system automatically separates code and data. This allows both systems programs such as editors and compilers as well as user code to be both shared and re-entrant. This method of memory utilization results in a more efficient and higher performance system.

System Accounting and Security Access Protection — The XENIX system has a complete security system for both system and file access. All data concerning system usage is also available for user accounting programs, and system tuning.

In addition to the operating system itself, the XENIX system includes all of the support software developed by Bell Laboratories that has been added to the UNIX system during the last decade. It includes:

- Microsoft implementation of the C language compiler
- Text editing and typesetting software
- Subroutine libraries
- Assembler and debugger
- System software development tools

COMPREHENSIVE SELF-TEST AND SYSTEM DIAGNOSTICS

In order to ensure correct system operation and rapid location of both hardware and software problems, the SYSTEM 86/330X is supplied with two levels of diagnostic routines:

SCT (System Confidence Test) — Automatically executed at power on and system reset, the SCT performs a GO-NO GO condition for the processors, memory and devices in the system.

SDT (System Diagnostic Test) — In the event that the SCT finds a system problem, the SDT may be executed by the user for a detailed analysis of the hardware status. This easy-to-use monitor program can perform tests on all hardware boards and mass storage peripherals in the system.

OTHER OPERATING SYSTEM SUPPORT

In addition to the XENIX operating system, the SYSTEM 86/330X also includes a preconfigured version of iRMX 86, Intel's high performance real-time operating system. A fully-configurable version of iRMX 86 is also available. Contact your Intel sales engineer or your Intel distributor for more information.

SPECIFICATIONS

Word Size

GENERAL PURPOSE PROCESSOR

Instruction — 8, 16, or 32 bits

Data — 8/16 bits

Instruction Cycle Time

250 nanoseconds for fastest executable instructions (assuming the instruction is in the queue).

750 nanoseconds for fastest executable instructions (assuming the instruction is not in the queue).

Memory Capacity

RAM — 384K bytes supplied with the base system. Memory may be expanded to 1 megabyte.

Interface

EIA Standard RS232C signals provided and supported.

Serial — 5 RS232C connectors configurable from 110 to 9.6K baud (asynchronous)

Parallel — A 50-pin parallel I/O port that is supplied with signal levels compatible with the industry standard Centronics interface printer (OEM supplied cable)

AC Requirements

5A @ 88 to 126 VAC, 60 Hz, single-phase (US only); 2.5A @ 176 to 252 VAC, 50 Hz, single-phase (Europe only); 5.5A @ 88 to 126 VAC, 50 Hz, single-phase (Japan). Maximum total power consumption 693W.

ORDERING INFORMATION

Part Number	Description
-------------	-------------

SYS 86/330AAX Kit	120V, 60 Hz (North American Version)
-------------------	---

SYS 86/330ABX Kit	220V, 50 Hz (European Version)
-------------------	-----------------------------------

SYS 86/330ACX Kit	100V, 50 Hz (Japanese Version)
-------------------	-----------------------------------

Product Safety Standards

The system is listed under UL standard 114 Safety of Electronic Data Processing Units and Systems. It is also certified by the Canadian Standards Association, standard C22.2 154-1975 Safety of Data Processing Equipment. The system complies with the International Electronics Commission standard IEC 435 Safety of Data Processing Equipment. The system also conforms to the applicable RFI/EMI requirements of VDE 0871/6.78, VDE 0875/6.77 and FCC rule 47 CFR part 15 subpart J Emission Limits for Computing Devices.

Environmental Requirements

OPERATING

Temperature — 15°C to 35°C

Relative Humidity — 20% to 80% non-condensing over the operating temperature range*

Vibration — 1.0g @ 10 to 55 Hz

*NOTE: The environmental combination of humidity and temperature together cannot exceed 26°C wet bulb.

NON-OPERATING

Temperature — -25°C to 60°C

Relative Humidity — 20% to 80% non-condensing

Vibration — 1.0g for 5 ms shock

Physical Characteristics

Width — 16.75 in. (42.55 cm)

Height — 12.25 in. (31.12 cm)

Depth — 23.00 in. (58.42 cm)

Weight — 80 pounds (34.02 kg)



SYSTEM 86/330A MICROCOMPUTER SYSTEM iRMX™ 86 OPERATING SYSTEM

- Open to VLSI extensions and a rapid upgrade path through standard VLSI hardware modules and interfaces
- High performance 16-bit iAPX 86/20 processor set (iSBC® 86/30 + iSBC® 337 boards)
- Full function iRMX™ 86 real-time, multitasking operating system
- Intel® resident languages include PL/M-86 and ASSEMBLER-86. Intel® PASCAL-86, FORTRAN-86, plus independent software vendor languages (BASIC, COBOL, C), also available
- MULTIBUS® system bus (IEEE-796) multiprocessor architecture
- Compact desk-top or rack-mount integrated microsystem
- 35MB Winchester and 1MB DS/DD 8" floppy for program, data storage and back-up
- 384KB of high speed RAM memory to execute multiple jobs and tasks
- Extensive self-test routines for reliable operation and simple fault isolation

The Intel SYSTEM 86/330A Microcomputer System is a comprehensive, real-time, integrated, 16-bit hardware and software package designed to give the OEM the fastest path to high performance VLSI. The system, which is based on standard Intel MULTIBUS system bus board level products, also incorporates the iRMX 86 VLSI operating system, state-of-the-art high-capacity mass storage and high-density RAM memory boards. In addition to the 8086 general purpose microprocessor, the system provides 3 to 5 times the numeric performance of low-end minicomputers through the use of the 8087 numeric data processor. The system's capabilities can be greatly expanded through the use of additional general purpose processors and intelligent I/O boards.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPL, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, iSXM, MULTICHANNEL, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

SYSTEM TECHNICAL DESCRIPTION

Hardware

DESIGNED FOR OEM OPEN SYSTEMS MARKET REQUIREMENT

The SYSTEM 86/300 family of microcomputer systems are designed to meet the emerging OEM systems market requirement for "open" systems. Open-systems requirements demand that the system be (1) open to easily absorb the next two generations of VLSI microcomputers, (2) open to instantly leverage off industry standard hardware modules and interfaces (such as MULTIBUS and SBX), (3) open to easily tap industry standard software from Intel and independent vendors, (4) open to industry standard data communication interconnections, and (5) open to allow the OEM to participate at any level of integration, systems, boards and components. The open SYSTEM 86/300 microcomputer family provides the OEM with instant access to VLSI technology, and the instant leverage and stability of software and hardware standards and documentation.

OPEN SYSTEMS ARCHITECTURE ALLOWS MULTIPLE LEVELS OF INTEGRATION

The entire SYSTEM 86/330A, both hardware and software, is built using standard, modular off-the-

shelf Intel board level products and system software. The system is designed so that the OEM can easily tailor a custom system through the use of individual Intel products needed to tailor a custom system. Through the use of industry standard modules and interfaces, the OEM can perform his own system integration without changing software code, thus lowering product cost.

PROCESSOR SECTION

The single board computer used in the SYSTEM 86/330A is the MULTIBUS-based iSBC 86/30 board. The central processor of the iSBC 86/30 board is the powerful 8 MHz 16-bit 8086 (5 MHz when used with the 8087 numeric co-processor). The architecture includes four 16-bit byte addressable registers, two 16-bit index registers, all accessed by a total of 24 operand addressing modes for complex data handling and flexible memory addressing.

The base SYSTEM 86/330A is supplied with 384KB of high speed RAM memory. There is 128KB of dual-ported RAM resident on the iSBC 86/30 processor board accessible to both the local and MULTIBUS system bus and can address up to one megabyte of main memory. The processor board includes an RS232C serial channel for the connection of a user supplied terminal. Cabling to the rear panel of the system is provided. Asynchronous baud rates of 110 to 19.2K are supported by the

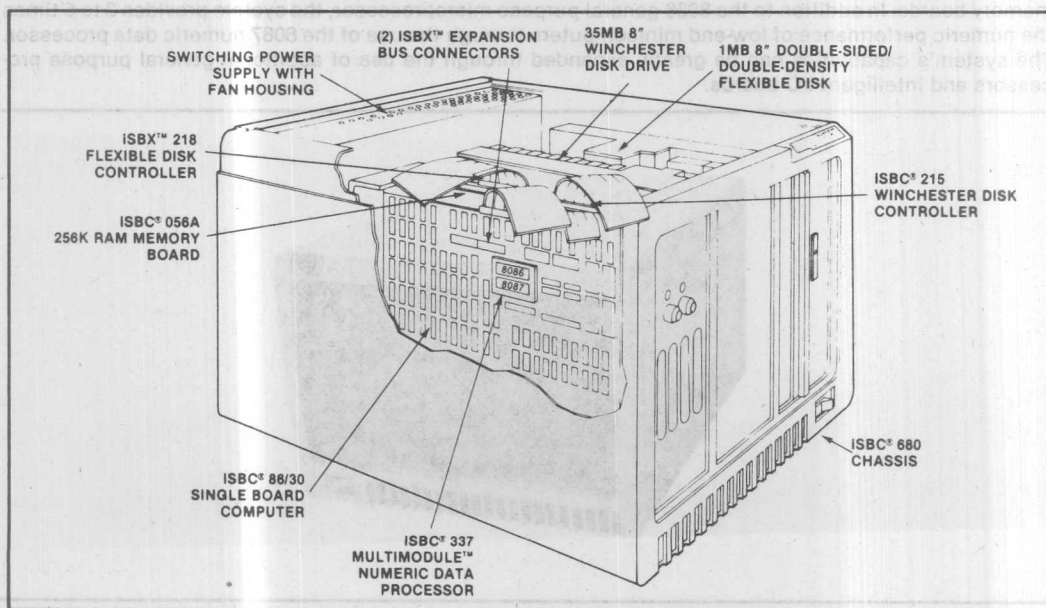


Figure 1. SYSTEM 86/330A Microcomputer System

system. A parallel I/O interface cabled to the back of the chassis is also supplied with the SYSTEM 86/330A. This 50-pin cable supports the industry standard Centronics printer signals.

NUMERIC PROCESSING EXTENSION

Also included with the base SYSTEM 86/330A is the iSBC 337 Numeric Data Processor. This MULTIMODULE board contains the Intel 8087 numeric co-processor. The co-processor interface between the 8087 and the 8086 host CPU provides a simple means of extending the instruction set of the system with over 60 additional numeric instructions supporting six additional data types. The data formats used in the SYSTEM 86/330A conform to the proposed IEEE floating point standard insuring highly accurate results. Dramatic numeric performance improvements are provided by the 8087 co-processor. For example a 50X improvement in the Whetstone benchmark over the standard 8086 processor is afforded by the 8086 numeric co-processor.

The 8087 arithmetic, logarithmic, transcendental and trigonometric instructions are fully supported by ASM-86, as well as other Intel higher level languages such as PL/M-86, PASCAL-86 and FORTRAN-86.

MEMORY EXPANSION

In addition to the 128KB of dual-ported RAM on the iSBC 86/30 processor board, the SYSTEM 86/330A also includes a 256KB memory expansion board. This brings the system RAM memory supplied with the base SYSTEM 86/330A to 384KB. The memory used is the latest technology 64K bit dynamic RAMs. Through the use of optional memory expansion boards, the total memory capacity of the SYSTEM 86/330A may be expanded to one megabyte.

MASS STORAGE

Winchester Disk Drive — The SYSTEM 86/330A contains a high performance 35MB (32MB formatted) 8" Winchester technology hard disk drive for program and data storage. The drive has an average access time of 43 ms and a transfer rate of 6.44 Mbts/sec.

Floppy Diskette Drive — A double-density/double-sided, 8", 1MB, floppy disk drive is included in the base system. This high-density floppy drive, which has an average access time and a transfer rate of 500Kbts/sec, can be used for both data storage and system backup.

Intelligent Controller — The SYSTEM 86/330A uses the intelligent, 8089-based iSBC 215 Winchester controller. This high performance interface contains firmware which is executed directly on the 8089 processor. This intelligent I/O processor coupled with an on-board RAM buffer off-loads a significant portion of disk I/O overhead from the host 8086 processor. In addition to the Winchester control firmware, there is also 8089 firmware resident on the iSBC 215 board to control the iSBX 218 floppy disk controller. The floppy controller plugs directly onto the iSBC 215 board via one of the two available iSBX connectors.

Mass storage expansion beyond the single Winchester and floppy diskette drives configured with the system is easily accomplished by the OEM through the use of daisy chain connectors on the back panel of the SYSTEM 86/330A chassis. In addition to the Winchester and floppy interfaces provided with the system, Intel also can provide a board level interface and iRMX 86 software for SMD compatible disk drives.

SYSTEM CHASSIS

The chassis used in the SYSTEM 86/330A is the compact iSBC 680 Multistore User System Package. This chassis may be used either as a desk-top unit or, with user supplied rack slides, as a rack-mount system. This 21.0" x 16.75" x 12.25" package houses up to six MULTIBUS cards (two available for expansion in the SYSTEM 86/330A). The 8" Winchester disk drive and 8" floppy disk drive are housed in the iSBC 680 package with simple slide-in, slide-out service access. In addition to providing the necessary voltages for the MULTIBUS cards, the efficient switching power supply also provides the 24 volts needed for the DC powered Winchester drive. The SYSTEM 86/330A has been designed to meet UL, CSA, FCC and VDE safety and EMI/RFI requirements. Supplied with the SYSTEM 86/330A, a 50-pin parallel connector, are connectors for the RS232C channel and additional floppy and Winchester disk drives. Cut-outs are also available for OEM supplied connectors.

System Software

As well as being the target system for OEM applications, the SYSTEM 86/330A can also be used for iRMX 86 software development.

VLSI OPERATING SYSTEM (iRMX™ 86)

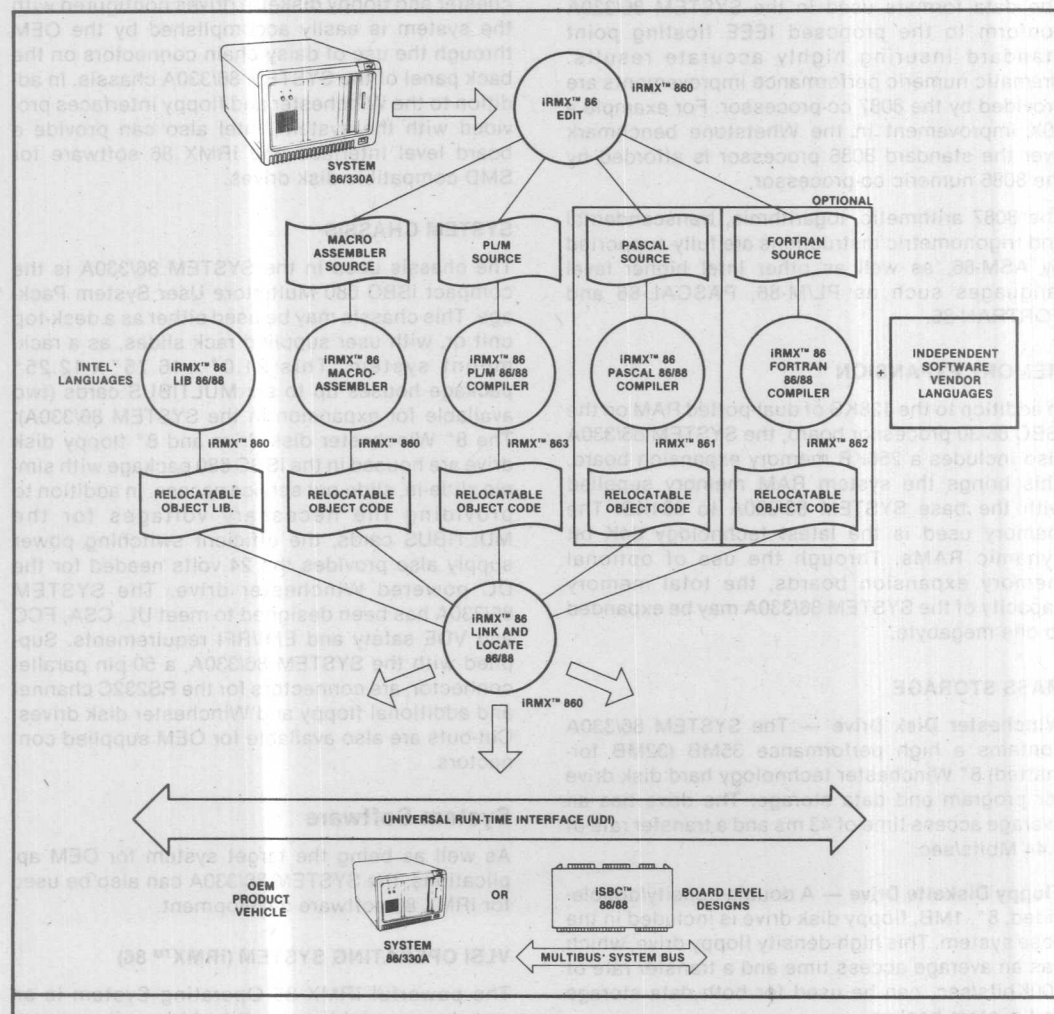
The powerful iRMX 86 Operating System is an easy-to-use, real-time multitasking software sys-

tem designed not only for the SYSTEM 86/330A, but for iAPX 86/88-based iSBC board and component level designs.

Services provided by the RAM-based iRMX 86 Operating System include facilities for executing programs concurrently, sharing resources and information, servicing asynchronous events, and interactively controlling system resources and utilities. In addition, the iRMX 86 Operating System provides all major real-time facilities, including priority-based system resource allocation, means for concurrently monitoring and controlling multiple external events, real-time clock control, interrupt management, and task dispatching. The iRMX

86 Operating System contains the following modules: an object-oriented Nucleus; Device Independent Basic and Extended I/O Systems; Terminal Handler; Bootstrap and Application Loaders; Human Interface with complete command line interpreter; and an interactive, object-oriented Debugger.

Because the modules and services provided by the operating system are user selectable, application specific operating systems can be created by iRMX 86 users. The iRMX 86 Operating System eliminates the need for custom operating system design, thereby reducing development time, cost, effort and risk.



IRMX™ UTILITIES PACKAGE (IRMX™ 860)

The iRMX Utilities Package consists of the following software:

IRMX™ 86 EDIT — The iRMX 86 EDIT program provides users with a powerful, sophisticated, line-oriented editing facility. EDIT delivers a range of capability suitable for novice users as well as advanced capabilities for sophisticated users. Its key features include a macro processor capable of creating and executing complex strings of commands, which ease the editing chore, as well as defining blocks of text which may be included anywhere in the text file. EDIT offers variable command sourcing, symbolic line numbering and reference by symbol. The facilities of EDIT allow users to create, maintain and manipulate extensive libraries of source code with minimal effort. For more information on iRMX 86 EDIT, see the EDIT Software Package data sheet (143883).

IRMX™ 86 LINK/LOCATE — The iRMX 86 LINK/LOCATE program connects object modules which have been individually compiled into a single, relocatable object module. The input object code may have been produced by any Object Module Format-compatible compiler. Output object modules may be recombined into larger object modules, allowing work from a large programming staff to be easily integrated into an application system.

IRMX™ 86 LIB — the iRMX 86 LIB "Library Manager" allows creation and maintenance of object module libraries. These libraries allow easy collection of related object code to reduce the overhead of maintaining many separate modules. Users may create new libraries, add and delete object modules, as well as list the contents of the library and their public symbols.

PL/M-86 (IRMX™ 863)

The PL/M-86 compiler provides users with a powerful, microcomputer-oriented system programming language. The PL/M-80 Language was introduced in 1976 by Intel. It was the first microcomputer-oriented, block structured, high-level language available. Since 1976, thousands of users, shipping over millions of microcomputer-based systems have generated their system software with PL/M-80 and PL/M-86.

PL/M-86 is a compatible superset of PL/M-80 which offers easy portability of software across the full range of microcomputers supplied by Intel. For more information about PL/M-86, see the PL/M 86/88 Software Package data sheet (402175).

FULL REALMATH SUPPORT

The iRMX 86 Languages support the REALMATH floating point standard. This allows users of all iRMX 86 languages to access the iAPX 86/20 Numeric Data Processor using the iSBC 337 MULTIMODULE board. These numeric processors offer over 100 times greater performance than comparable software-implemented algorithms, and reduce the system memory requirements by at least 16KB. The REALMATH standard (proposed IEEE standard) provides universal consistency in results of numeric computations. The iRMX 86 Languages provide efficient object code generation and access to the highest performance floating point package available on microcomputers.

COMPREHENSIVE SELF-TEST AND SYSTEM DIAGNOSTICS

In order to insure correct system operation and rapid location of both hardware and software problems, the SYSTEM 86/330A is supplied with three levels of diagnostic routines:

SCT (System Confidence Test) — Automatically executed at power on and system reset, the SCT performs a GO-NO GO condition for the processors, memory and devices in the system.

SDT (System Diagnostic Test) — In the event that the SCT finds a system problem, the SDT may be executed by the user for a detailed analysis of the hardware status. This easy-to-use monitor program can perform tests on all hardware boards in the system and mass storage peripherals.

SAT (System Analysis Test) — This diagnostic tool tests the integration of the hardware and software at the system level. This helps to isolate intermittent failures by running a load stress test on both hardware and software.

SYSTEM DEBUGGING TOOLS

The iRMX 86 Operating System provides a comprehensive tool for interactive software debugging. The Debugger has two capabilities that greatly simplify the process of debugging a multitasking system. First, the Debugger allows users to debug several tasks while the balance of the application system continues to run in real-time. Second, the Debugger allows programmers to interactively view and modify system constructs as well as the system RAM and CPU registers. The Debugger is structured to enable system designers to track system-wide problems easily. It can also remain in the final OEM application as a continuous maintenance tool.

OEM SOFTWARE LICENSING

The Intel software products listed above require the signing of an Intel Master Software License Agreement. All products include 1 year of update service. Software is shipped on floppy media in two object forms: 1) a ready-to-run, fully configured system, and 2) a configurable version of all software products.

OPTIONAL INTEL® SOFTWARE SUPPORT

In addition to ASM-86 and PL/M-86 included in the base SYSTEM 86/330A, the following Intel languages are available for use on the system:

PASCAL-86 (iRMX™ 861) — The PASCAL-86 compiler provides a strict implementation of the proposed ISO language standard. All source programs are validated by the compiler to ensure its conformance to the standard. Many extensions to the language are available which allow PASCAL programs to be written specifically for microcomputers. Separate module compilation and iAPX 86/20, 88/20 Numeric Data Processor support are a few of its many features. The ISO standard "source evaluator" can be switched off to accept these extensions. For more information on iRMX 86 PASCAL features, see the PASCAL 86/88 Software Package data sheet (121680).

FORTRAN-86 (iRMX™ 862) — The iRMX 86 FORTRAN compiler provides users total compatibility with existing FORTRAN 86 language-generated code, plus many new language features provided by the FORTRAN 77 language standard. These new features offer FORTRAN programmers many new capabilities, including "IF-THEN-ELSE", random access I/O and character variables. For a more detailed explanation of iRMX 86 FORTRAN, see the FORTRAN 86/88 Software Package data sheet (400630).

iMMX 800 MULTIBUS® Message Exchange — The advanced design of the MULTIBUS system bus coupled with the iMMX 800 software package allows the SYSTEM 86/330A to easily support additional 8 and/or 16-bit Intel single board computers. This powerful option enables OEMs to increase

the SYSTEM 86/330A's general purpose processing power with boards such as the iSBC 86/05 (8 MHz 8086) and iSBC 88/25 (5 MHz 8088). Intelligent, high performance microprocessor-based boards such as the Ethernet* communications controller (iSBC 550 board), serial communications controller (iSBC 544 board) and analog measurement and control computer (iSBC 88/40 board) can greatly expand the capabilities and processing power of the SYSTEM 86/330A.

Independent Software Vendor Support — Through the use of the standard UDI (Universal Development Interface) a wide variety of language products, both compilers and interpreters, are now available for use on iRMX 86 from independent software vendors. COBOL, BASIC and C are a few examples of the available languages. Contact your Intel sales representative for more information.

System Options

HARDWARE

Any Intel MULTIBUS board may be installed by the OEM into the two available expansion slots in the system chassis.

SOFTWARE

Operating Systems — In addition to the high performance iRMX 86 real-time operating system, a version of the SYSTEM 86/330A is also available with the interactive XENIX† (UNIX‡ V7) operating system. See your Intel sales representative or distributor for more details.

Languages — Optional language products are available from both Intel and independent software vendors. Operating system support for additional 8085, 8088 and 8086-based processor boards is available through the use of iRMX 80, iRMX 88 and iRMX 86 software. Multiprocessor software support, iMMX 800, is also available for the SYSTEM 86/330A.

* ETHERNET is a trademark of Xerox Corp.

† UNIX is a trademark of Bell Laboratories.

‡ XENIX is a trademark of Microsoft Corp.

SPECIFICATIONS

Word Size

GENERAL PURPOSE PROCESSOR

Instruction — 8, 16, or 32 bits

Data — 8/16 bits

NUMERIC CO-PROCESSOR

Data — 80 bits

Instruction Cycle Time

400 nanoseconds for fastest executable instructions (assuming 5 MHz clock rate and the instruction is in the queue).

1.0 microseconds for fastest executable instructions (assuming 5 MHz clock rate and the instruction is not in the queue).

Memory Capacity

RAM — 384K bytes supplied with the base system. Memory may be expanded to 1 megabyte.

Interface

EIA Standard RS232C signals provided and supported.

Serial — Configurable from 110 to 19.2K baud (asynchronous)

Parallel — A parallel I/O port that is supplied with signal levels compatible with the industry standard Centronics interface printer (OEM supplied cable)

iSBX Bus Expansion — Two 16-bit iSBX connectors available for VLSI MULTIMODULE expansion

AC Requirements

6.5A @ 88 to 126 VAC, 60 Hz, single-phase (US only); 3.25A @ 176 to 252 VAC, 50 Hz, single-phase (Europe only); 6.5A @ 88 to 126 VAC, 50 Hz, single-phase (Japan). Maximum total power consumption 693W.

Product Safety Standards

The system is listed under UL standard 114 Safety of Electronic Data Processing Units and Systems. It is also certified by the Canadian Standards Association, standard C22.2 154-1975 Safety of Data Processing Equipment. The system complies with the International Electronics Commission standard IEC 435 Safety of Data Processing Equipment. The system also conforms to the applicable RFI/EMI requirements of FCC rule 47 CFR part 15 subpart J Emission Limits for Computing Devices.

Environmental Requirements

OPERATING

Temperature and Relative Humidity — 15°C to 35°C dry bulb and 20% relative humidity to 80% relative humidity non-condensing subject to a maximum wet bulb temperature of 26°C. These conditions translate to a maximum relative humidity of 80% non-condensing up to 28.8°C which falls to a maximum relative humidity of 50% at 35°C

Altitude — Sea level to 10,000 feet

Vibration — 0.0014 inches peak-to-peak, 5-25 Hz; 0.0007 inches peak-to-peak, 25-55 Hz; 0.3g 0-to-peak, 55-300 Hz

Shock — 1.0g for 11 ms

NON-OPERATING

Temperature — 25°C to 60°C

Relative Humidity — 20% to 80% non-condensing

Altitude — Sea level to 40,000 feet

Vibration — 0.008 inches peak-to-peak, 5-25 Hz; 0.004 inches peak-to-peak, 25-55 Hz; 2.0g 0-to-peak, 55-300 Hz

Shock — 15g for 11 ms

Physical Characteristics

Width — 16.75 in. (42.55 cm)

Height — 12.25 in. (31.12 cm)

Depth — 21.00 in. (53.34 cm)

Weight — 75 pounds (34.02 kg)

Extensive System Documentation

The SYSTEM 86/330A is shipped with six documentation binders containing over 28 in-depth manuals on all aspects of hardware and software operation. A system installation and maintenance manual, in addition to a system overview manual, is also included.

ORDERING INFORMATION

Part Number	Description
-------------	-------------

SYS 86/330AAR Kit	120V 60 Hz (North American Version)
-------------------	--

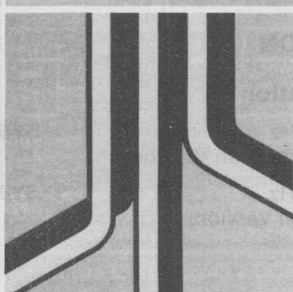
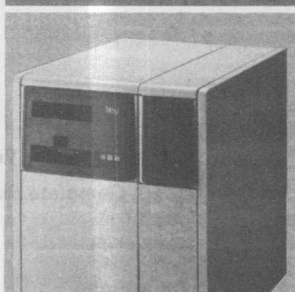
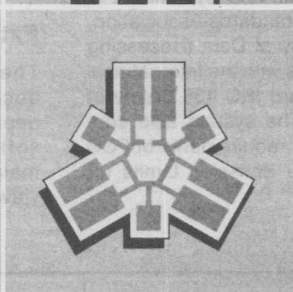
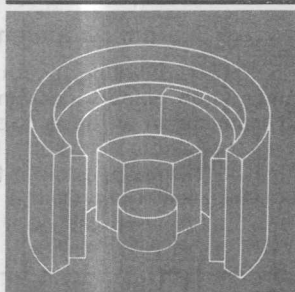
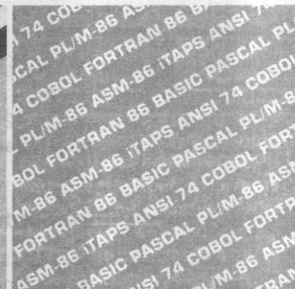
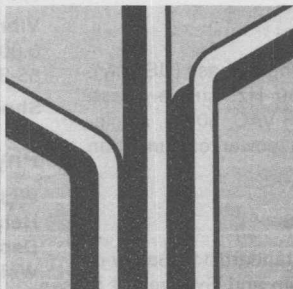
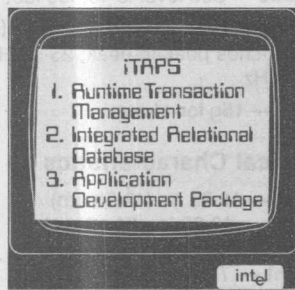
SYS 86/330ABR Kit	220V 50 Hz (European Version)
-------------------	----------------------------------

SYS 86/330ACR Kit	100V 50 Hz (Japanese Version)
-------------------	----------------------------------

SYS 86/330A Doc	Complete manual set
-----------------	---------------------

iTPS 86/445 Transaction Processing System

- A high-performance, expandable commercial microcomputer system, supporting up to 16 users
- A streamlined execution environment for database and transaction-oriented applications
- A member of the Intel iTPS family of compatible systems
- Extensive communications capabilities including 3270 SNA, 3270 Bisync and 2780/3780 RJE
- An "Open System" designed for growth to future VLSI technology
- Fully supported by Intel's worldwide support and service organization



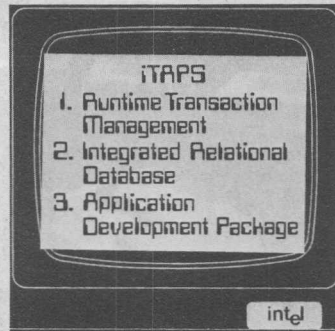
FUNCTIONAL DESCRIPTION

The iTPS 86/445 Transaction Processing System is a high-performance, multiuser system specifically designed for use in commercial transaction processing environments. It provides mainframe calibre capabilities in an economical, high-quality package of hardware and software. The excellent price-performance ratio of the iTPS 86/445, coupled with the flexibility with which it can be configured, make it the ideal vehicle for delivering cost effective solutions to a wide spectrum of commercial applications.

The iTPS 86/445 central system is housed in an attractive, table-height, free-standing unit. This contains the Intel 8086-based central processor, up to 1 megabyte of error corrected memory, from 35 to 336MB of disk storage, a 0.6MB diskette drive, and communications facilities for from 1-16 RS-232 terminals and one or more host links. Peripherals available with the system include terminals, a printer, and an 11.7MB 1/4-inch magnetic cartridge tape.

The iTPS software includes a powerful multiuser operating system, a unique software subsystem for developing and executing on-line applications, support for all the popular higher level languages, and a full complement of utilities.

The iTPS 86/445 is a high end member of the Intel iTPS family of transaction processing systems which offers a wider range of storage and communications capacities than other members of the family. Thus applications written for the iTPS 86/445, for example, will run without change on the iTPS 86/435 given equivalent resources. To the applications developer, this means software can be developed once and then be readily ported between large and small systems.



ITAPS APPLICATION ENVIRONMENT

The iTPS 86/445 can dramatically increase programmer productivity and streamline application execution with a powerful interactive software system called iTAPS—the Intel Terminal Application Processing System. iTAPS is a comprehensive transaction processing and database management software system which runs as a subsystem on top of the iTPS executive. The elements of iTAPS include a complete transaction monitor, an integrated relational database manager, an on-line database inquiry and update facility, an English-like query capability, five levels of security, and built-in recovery and restart mechanisms.

By providing a non-procedural, interactive approach to developing applications, iTAPS significantly reduces the time and effort required to design, implement, and maintain on-line applications. This allows the developer to deliver quality applications which are optimized for end-user execution performance yet can be easily customized for individual users' needs.

Applications are produced using an interactive development facility which generates internal iTAPS tables that define data structures, menu interfaces, and transaction flow. When the application is executed, these tables control how built-in iTAPS facilities are invoked and how data is accessed. In this way, many applications can be developed using just the built-in facilities provided by iTAPS, while more complex or specialized applications can be accommodated by adding user programs written in COBOL and PASCAL to the standard iTAPS modules. In addition, use of iTAPS ensures that the applications developer has the freedom to customize applications to match individual user requirements without having to perform major surgery on his software.

In the run-time environment, iTAPS' transaction processing monitor controls the overall execution of the application and provides multiple levels of access control for both programs and data. It also furnishes recovery and restart facilities to ensure the integrity of the content of databases. iTAPS' integrated relational database manager provides efficient, synchronized, shared access to the applications information and supports an English-like casual user query and update facility. This interactive facility allows casual users to access and update databases and to create and format their own reports using simple, free format commands.

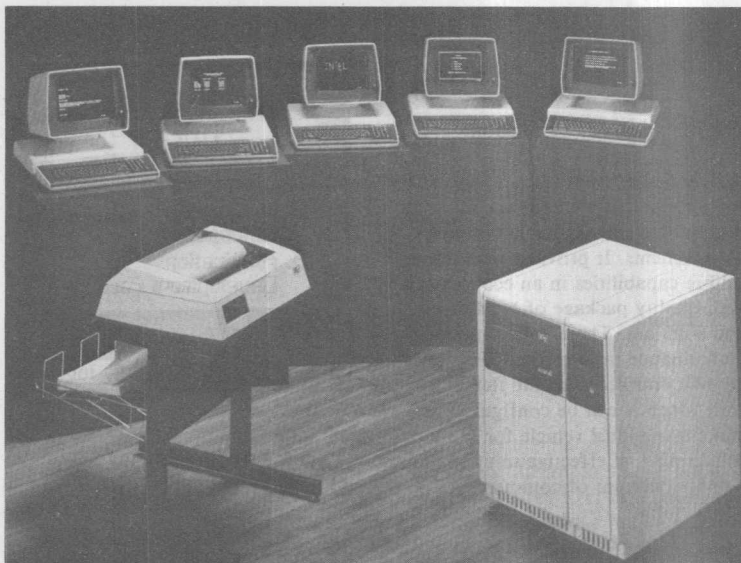
ITPS SYSTEMS SOFTWARE

The iTPS 86/445 provides users with a broad selection of the most popular programming languages for developing commercial applications. Comprehensive support is provided for ANSI 74 COBOL, ISO PASCAL, FORTRAN-77, BASIC, PL/M, and a macro assembler. This support includes extensive documentation, powerful debugging tools, and a full screen editor which enables programmers to easily create and update their source programs. Support utilities enable the user to combine separately compiled/assembled programs and execute them as a single entity. The iTPS high-level languages are all very compatible with mainframe implementations, ensuring minimal effort to convert existing applications onto the 86/445.



ITPS OPERATING SYSTEM

The operating system for the iTPS is a commercially enhanced version of the mature, high performance, real-time iRMX™ 86 executive. Known as iRMX 86C, it is a comprehensive multiuser, multiprogramming operating system which includes the capability to share files and programs between independent users. It is designed for use in a wide variety of high throughput applications, especially transaction-oriented processing.



One of the most significant features of the operating system is the Re-entrant Program Manager (RPM). This enables multiple users to share single copies of programs, and to dynamically page portions of large programs or data sets in and out of system memory. This system also provides SIOS, the Shared I/O System, which allows multiple users to share files. SIOS includes a dynamic record-level locking facility to ensure data integrity while multiple users are concurrently accessing and updating data in the same file.

ITPS 86/445 CENTRAL SYSTEM

The iTPS 86/445 applications processor is based on the Intel 8086, which is by far the world's most widely used 16-bit microprocessor. The powerful 8086 instruction set is ideal for use in high-performance multiprogramming environments, and has many features which optimize the performance of higher level languages. An RS-232C remote diagnostic port on the processor provides a facility to exercise and monitor the system remotely in diagnostic mode.

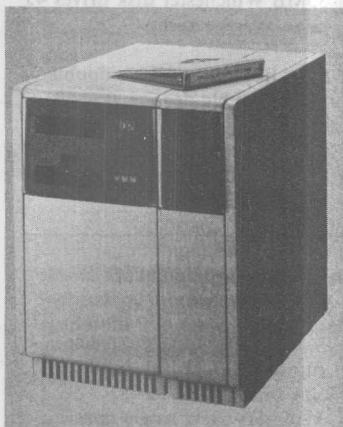
As with all of Intel's "Open Systems," the iTPS 86/445 uses the IEEE-796 standard Multibus, ensuring that future VLSI can be easily incorporated with the system. This also provides an easy path for adding custom or third-party supplied Multibus-based features to the system. Depending on the system configuration, as many as 7 open multibus board slots are available for such extensions.

The iTPS 86/445 central system includes up to one megabyte of error corrected main memory, from 35 to 336MB of Winchester disk storage, and a 0.6MB, 5-inch double-sided double-density diskette drive for removable storage. There is also provision for an optional 1/4-inch start/stop magnetic cartridge tape drive. In addition to a Centronics-compatible printer interface, the base system accommodates the attachment of up to four terminals via an intelligent communications controller. Additional controller modules can be added to allow up to 16 terminals to be configured.

MASS STORAGE DEVICES

The iTPS 86/445 offers a choice of two models of 8-inch Winchester disk drives. The high-performance 84MB capacity drive has an average access time of 20 msec and a transfer rate of 1.2MB/sec. The 35MB capacity drive has an average access time of 45 msec and a transfer rate of 0.8MB/sec. A total of four drives, of the same type, can be supported by the system.

For archival and back-up purposes, the iTPS 86/445 provides an optional 1/4-inch magnetic tape cartridge drive. This has a formatted capacity of 11.7M bytes, a transfer rate of 192K bits/sec, and uses a removable tape cartridge.



COMMUNICATIONS

The iTPS 86/445 provides a versatile set of communications facilities to support a wide variety of distributed processing environments. The system supports up to 16 RS-232C communications lines, operating in asynchronous mode at up to 19.2K bits/sec. For communication with remote host

systems, support is offered for 3270 communications emulation in both Bisync and SDLC/SNA modes; and for 2780/3780 (Bisync) communications emulation. Ethernet* connection can be accomplished by use of the iSBC®-551 controller module, and the customer can incorporate other specialized communications interfaces by using the open MULTIBUS® slots which are available in the systems enclosure.



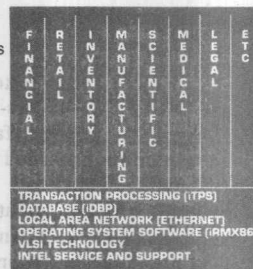
TERMINALS

The iTPS terminal workstation is an intelligent, full function, ergonomically superior terminal. The iTPS terminal is human engineered to provide the maximum in ease-of-use and efficiency. The screen rotates and tilts, and the detached keyboard has sculptured key caps in a typewriter style layout, plus 16 function keys and a 14-key numeric pad. A local RS-232C

*Ethernet is a trademark of Xerox Corporation

SYSTEMS
INTEGRATORS
SOLUTION
LEVEL

INTEL'S
BUILDING
BLOCKS



interface enables the customer to attach a local printer to the terminal for hard copy of screen images. The terminal has a full array of intelligent features, including full cursor addressability, full editing, protected fields, reverse video, blink, and underline.

The use of silicon software in the iTPS terminal distributes the editing of iTPS input data from the iTPS central processor down to the local device. This microcode can significantly enhance the performance of the iTPS 86/445 for iTPS applications by offloading the editing function

from the central processor, yet does not limit the device's use in non-iTPS mode. The feature also enhances operator performance by notifying the operator of input errors as they are entered.

PRINTER

The iTPS printer is attached to the iTPS 86/445 via a Centronics-compatible parallel I/O interface. This 200-character-per-second printer combines quality performance with quiet operation and excellent reliability. It can print both 10 and 16.5 characters/inch, handles up to 6-part forms, and has vertical spacing of 6 and 8 lines per inch. The printer also has excellent built-in diagnostics for easy maintenance.



WORLDWIDE SERVICE

The iTPS System, terminals, and printers are fully supported by Intel's worldwide Product Service organization. Initial system installation is also performed by Intel's trained customer service engineers and is included in the system's price.

SPECIFICATIONS

Word Size

Instruction—8, 16 or 32 bits
Data—8 or 16 bits

Instruction Cycle Time

250 nanoseconds for fastest executable instructions (assuming instruction is in the queue)
750 nanoseconds for fastest executable instructions (assuming instruction is not in the queue)

Memory Capacity

RAM—640K bytes are supplied with the base system, may be expanded to one megabyte. All memory has ECC.

Processor and Peripheral Cabinet

Contains: 5 1/4" floppy disk drive, 8" disk drive(s), optional 1/4" tape cartridge drive, power supplies, fans, MULTIBUS® cardcage with up to 7 slots available for custom use.

Dimensions: 29.5" high, 24.5" wide, 30" deep

Processor Interfaces

Serial—1 RS-232C from 110 to 19.2K baud, asynchronous
Parallel—1 parallel I/O port with Centronics printer interface

Communications

Serial—16 RS-232C ports, configurable from 110 to 19.2 baud, asynchronous
BISYNC—2780/3780 RJE emulation

—3271 emulation
SDLC—3274/76 SNA emulation
Ethernet: iSBC-550 controller

Mass Storage

1–4 84MB Winchester disk drives
20 msec average access or 1–4 35MB Winchester disk drives 45 msec average access

1 5 1/4" floppy disk drive double-sided, double-density, 0.6MB capacity

1 1/4" cartridge tape drive, 11.7MB formatted capacity

ENVIRONMENTAL REQUIREMENTS

Operating

Temperature—15 °C to 32 °C
Humidity—20% to 80% non-condensing over the operating temperature range, with a 26 °C wet bulb temperature limit
Altitude—Sea level to 6000 feet
Vibration—Less than 0.2G @ 3 to 60Hz

Non-operating

Temperature—25 °C to 60 °C
Humidity—10% to 80% (non-condensation)
Altitude—Less than 40,000 feet
Vibration—Less than 0.4G @ 3 to 60Hz
—Less than 3G, non-cyclic during shipping

AC REQUIREMENTS

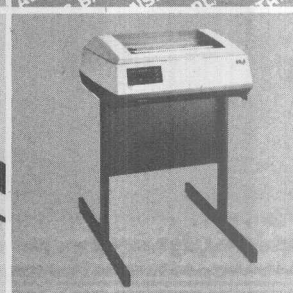
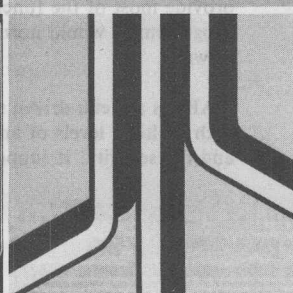
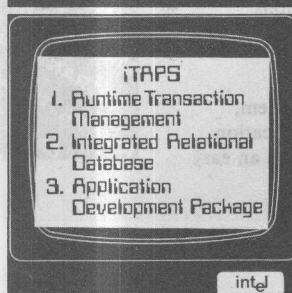
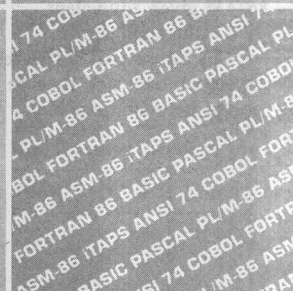
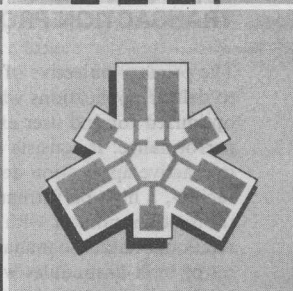
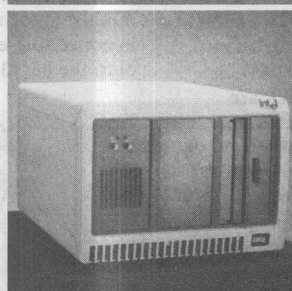
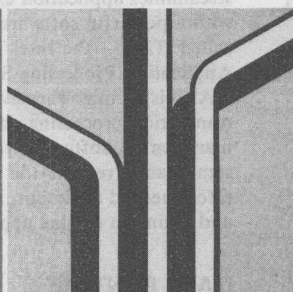
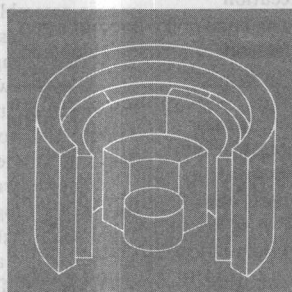
12.5A (max) @ 90 to 130 VAC, 50/60Hz single-phase (up to 2 disk drives only)
9.2A (max) @ 95 to 255 VAC, 50/60Hz single-phase

PRODUCT SAFETY STANDARDS

The system is listed under UL Standard 478 Safety of Electronic Data Processing Units and Systems. It is also certified by the Canadian Standards Association, Standard C22.2 154-1975 Safety of Data Processing Equipment. The system also conforms to the applicable RFI requirements of VDE 871/6.78 class A, VDE 875 limit N and FCC rule 47, CFR part 15 subpart J, Emissions Limits for Computing Devices.

ITPS 86/433 Transaction Processing System

- A major advancement in integrating mainframe functionality with the microcomputer
- A low cost, multiuser desktop workstation
- Another member of Intel's iTPS family of transaction processing systems
- A streamlined execution environment for database and transaction applications
- An "Open System" for future growth to new VLSI
- Fully supported by Intel's worldwide support and service organization



©INTEL CORPORATION, 1983

FEBRUARY 1983
ORDER NUMBER: 210913-001

FUNCTIONAL DESCRIPTION

The iTPS 86/435 Transaction Processing System brings mainframe calibre on-line processing capabilities and a highly attractive price-performance ratio to the small business environment. It provides a high-quality, effective, and economical vehicle for delivering transaction-oriented application systems to end user environments.

The iTPS 86/435 is a desktop system designed for efficient execution of interactive commercial applications. Its hardware includes a central processor with 768K bytes of memory, a 35MB disk unit, a one megabyte diskette, optional terminals and printers, and powerful application development software. It supports one to four users in a multiuser, multiprogramming environment using a commercially augmented version of the iRMX™ 86 Operating System.

The iTPS 86/435 is a mid-range member of the Intel iTPS family. Thus, applications written for other members of the iTPS family, e.g., the 86/445, will run without change on the 86/435 given equivalent resources. To the applications developer, this means that software can be developed once and then be readily ported between large and small systems.



ITAPS APPLICATION DEVELOPMENT PACKAGE

The iTPS 86/435 increases programmer productivity and streamlines application execution with a powerful software package called iTPS—the Intel Terminal Application Processing System. iTPS is a comprehensive transaction processing and database management software system which significantly reduces the time and effort needed to design, implement, and maintain on-line applications.

ITAPS RUNTIME TRANSACTION PROCESSING

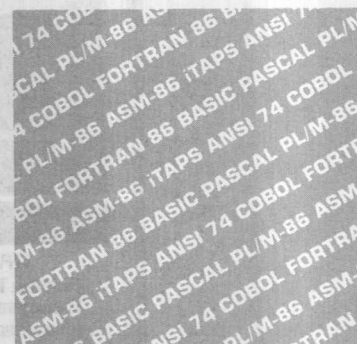
The primary objective of iTPS is to deliver applications which are optimized for end user execution performance. It consists of an interactive application development facility, a high-performance transaction monitor with integrated relational database manager, and a set of built-in modules which provide most of the functions programmers would normally have to write.

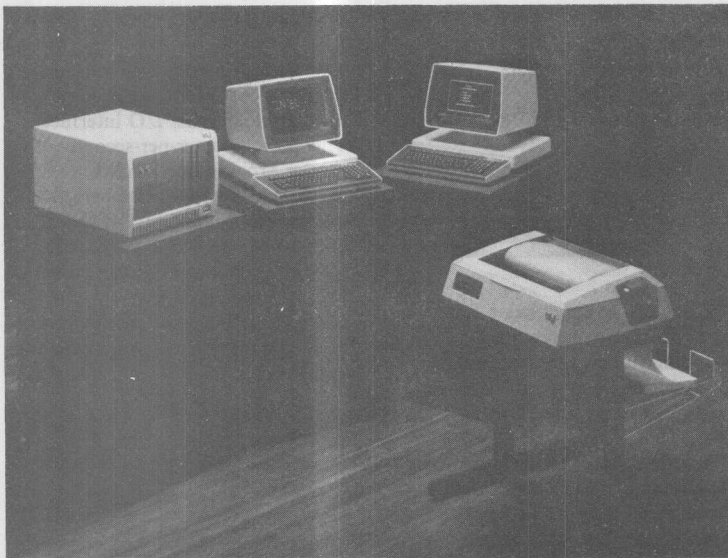
iTPS is a menu driven system, with multiple levels of application and file security. It supports an easy

to use on-line database query and update facility, which allows unsophisticated users to access databases and retrieve and format their own reports using simple English language commands. Full database recovery and restart facilities are integrated with iTPS to ensure the integrity of all application systems.

ITPS LANGUAGES

The iTPS 86/435 provides users with a broad selection of the most popular programming languages for developing commercial applications. Comprehensive support is provided for ANSI 74 COBOL, FORTRAN, PASCAL, BASIC, PL/M, and a macro assembler. This support includes extensive documentation, powerful debugging tools, and full screen editor which enables programmers to easily create and update their source programs. Support utilities enable the user to combine separately compiled/assembled programs and execute them as a single entity. The iTPS languages are all highly compatible with the mainframe implementations, ensuring minimal effort in converting existing applications onto the 86/435.





iTPS iRMX™ 86C OPERATING SYSTEM

iRMX 86C is a multiuser, multiprogramming operating system with a shared file system and extensive utilities. It is designed for use in a wide variety of applications, especially transaction-oriented processing. iRMX 86C is a mainframe calibre executive, with many features normally associated only with large systems. One of the most significant features of the operating system is the Re-entrant Program Manager (RPM), which enables multiple users to share a single copy of a program. The system also includes SIOS, the Shared I/O System, which allows multiple users to share files, with dynamic record-level locking to ensure integrity.

iTPS 86/435 CENTRAL SYSTEM

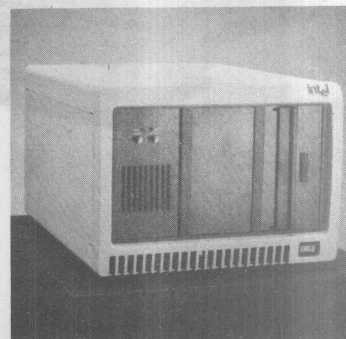
The iTPS 86/435 applications processor is based on the Intel 8086, which is by far the world's most widely used 16-bit microprocessor. The powerful 8086 instruction set is ideal for use in high-performance multiprogramming environments, and it has many features which optimize the performance of higher level languages. An RS 232C remote diagnostic port on the processor provides a facility to exercise and monitor the system remotely in diagnostic mode.

As with all of Intel's "Open Systems," the iTPS 86/435 uses the IEEE-796 standard MULTIBUS®, ensuring that future Intel VLSI can be easily incorporated with the system. This also enables systems integrators and others to add custom or third-party supplied MULTIBUS-based features to the

system. An open MULTIBUS-compatible slot is included in the 86/435 for this purpose. The iTPS 86/435 Central System includes 768KB of main memory, a 35MB Winchester disk drive for primary on-line storage, and a 1MB 8 inch double-sided double-density floppy disk for removable storage. In addition to a Centronics-compatible printer interface, the system accommodates the attachment of up to four terminals via an intelligent communications controller.

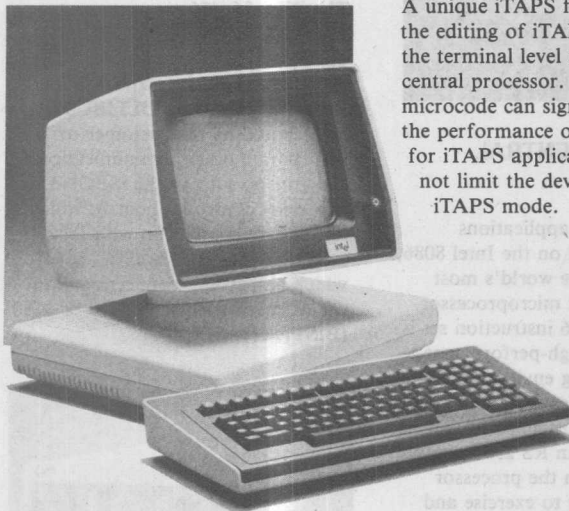
COMMUNICATIONS

The iTPS 86/435 supports up to four RS 232C communications lines, operating in asynchronous mode at up to 19.2K bits/sec. In addition, the system's open MULTIBUS slot can be used by the customer to incorporate other communication controllers, such as the iSBC-551 (Ethernet controller board), which allows interconnection with local area networks, or the iSBC-88/45, which allows interconnection with remote networks using 3270 or 3780 (BISYNC or SDLC).



TERMINALS

The iTPS terminal workstation is a very attractive, full function ergonomically superior terminal. Up to four of these can be attached to the iTPS 86/435. The iTPS terminal is human engineered to provide the maximum in ease-of-use and efficiency. The screen rotates and tilts, and the detached keyboard has sculptured key caps in a typewriter style layout, plus 16 function keys and a 14-key numeric pad. A local RS 232C interface enables a user-supplied printer to be attached to the terminal for hard copy of screen



SYSTEMS
INTEGRATORS
SOLUTION
LEVEL

INTEL'S
BUILDING
BLOCKS

FINANCIAL	RETAIL	INVENTORY	MANUFACTURING	SCIENTIFIC	MEDICAL	LEGAL	ETC
TRANSACTION PROCESSING (iTPS) DATABASE (iDBP) LOCAL AREA NETWORK (ETHERNET) OPERATING SYSTEM SOFTWARE (iRMX86) VLSI TECHNOLOGY INTEL SERVICE AND SUPPORT							

images. The terminal has a full array of intelligent features, including full cursor addressability, full editing, protected fields, reverse video, blink and underline.

A unique iTAPS feature distributes the editing of iTAPS input data to the terminal level from the iTPS central processor. This local microcode can significantly enhance the performance of the iTPS 86/435 for iTAPS applications, and does not limit the device's use in non-iTAPS mode.

PRINTER

The iTPS printer can be attached to the iTPS 86/435 via a Centronics-compatible parallel I/O interface. This 200-character-per-second printer provides top quality performance with quiet operation and excellent reliability. It can print both 10 and 16.5 characters/inch, handles up to 6-part forms, and has vertical spacing of 6 and 8 lines per inch. The printer has excellent built-in diagnostics for easy maintenance.



WORLDWIDE SERVICE

The iTPS System, terminals, and printers are all fully supported by Intel's worldwide Product Service organization. Initial system installation is also performed by Intel's trained customer service engineers and is included in the system's price.

SPECIFICATIONS

Word Size

Instruction— 8, 16 and 32 bits

Data— 8/16 bits

Instruction Cycle Time

Fastest executable instructions

— instruction in queue, 400 nanosec

— instruction not in queue, 1.0 microsec

Memory Capacity

768K bytes supplied with system

Interfaces

Serial — 1 RS 232C from 110 to 19.2K baud, asynchronous

Parallel — 1 parallel I/O port with Centronics printer interface

Communications

4 RS 232C ports, configurable from 110 to 19.2K baud, asynchronous

Mass Storage

1 8" 35MB Winchester technology disk drive

1 8" 1MB double-sided, double-density floppy disk drive

AC REQUIREMENTS

5-A at 88 to 126 VAC 60 HZ, single phase (U.S.)

2.5-A at 176 to 252 VAC 50 HZ, single phase (Europe)

5-A at 85 to 110 VAC 60 HZ, single phase (Japan)

Maximum total power consumption 550W

PRODUCT SAFETY STANDARDS

The system is listed under UL standard 114 Safety of Electronic Data Processing Units and Systems. It is also certified by the Canadian Standards Association, standard C22.2 154-1975 Safety of Data Processing Equipment. The system complies with the International Electronics Commission standard IEC 435 Safety of Data Processing Equipment. The system also conforms to the applicable RFI/EMI requirements for VDE 0871/6.78 and FCC rule CFR part 15 subpart J Emission Limits for Computing Devices.

ENVIRONMENTAL REQUIREMENTS

Operating

Temperature— 15 °C to 35 °C

Relative Humidity— 20% to 80% non-condensing over the operating temperature range*

Altitude— Sea level to 6000 feet

Vibration— 1.0g @ 10 to 55 HZ

*Note: The environment combination of humidity and temperature together cannot exceed 26 °C wet bulb.

Non-Operating

Temperature— -25 ° to 60 °C

Relative Humidity— 20% to 80% non-condensing

Altitude— Sea level to 12,000 feet

Vibration— 1.0g for 5ms shock

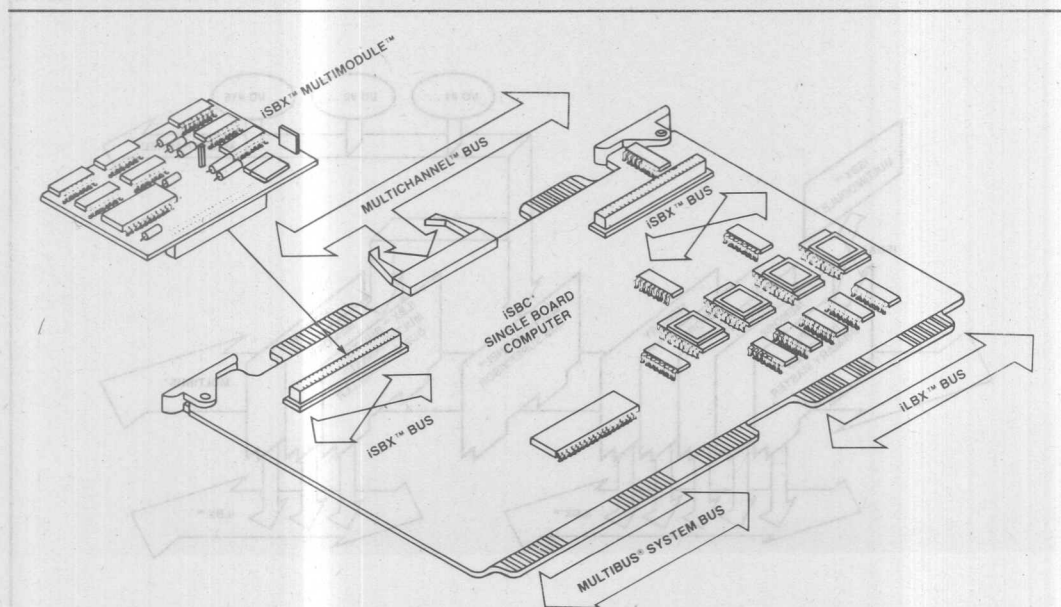
Single Board Computers

3

MULTIBUS® SYSTEM BUS

- IEEE 796 industry standard system bus
- Supports multiple processor systems with multi-master bus structure
- 8-bit and 16-bit devices share the same MULTIBUS® system resources
- Foundation of Intel's Total System Architecture: MULTIBUS®, iLBX™, MULTICHANNEL™ and iSBX™ busses
- 16 Mbyte addressing capability
- Bus bandwidth of up to 10 megabytes per second
- Supported by a complete family of single board computers, memory, digital and analog I/O, peripheral controllers, graphics and speech recognition, packaging and software
- Supported by over 150 vendors providing over 1000 compatible products

The MULTIBUS® System bus is one of a family of standard bus structures resident within Intel's total system architecture. The MULTIBUS interface is a general purpose system bus structure containing all the necessary signal lines to allow various system components to interact with one another. This device interaction is built upon the master-slave concept. The "handshaking" between master and slave devices allows modules of different speeds to use the MULTIBUS interface and allows data rates of up to 5 million transfers per second. The MULTIBUS system bus can support multiple master devices (16) on a 18 inch backplane and can directly address up to 16 megabytes of memory. As a non-proprietary, standard system bus, the MULTIBUS interface has become the most prominent 8/16-bit microcomputer system bus in the industry with over 150 vendors supplying over 1000 MULTIBUS compatible products. Its success as the industry standard has been reinforced by adoption of the MULTIBUS specification by the Institute of Electrical and Electronic Engineers — (IEEE 796 System Backplane Bus). MULTIBUS-based systems have been designed into applications, such as, industrial automation and control, office systems and word processing, graphics systems and CAD/CAM, telecommunications systems and distributed processing.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBX, iSBX, iSBX, iSBX, iSBX, MULTIBUS, Multichannel and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

FUNCTIONAL DESCRIPTION

Architectural Overview

The MULTIBUS® system bus is the physical framework and the conceptual foundation of Intel's total system architecture. It is a general purpose system bus used in conjunction with the single board computer concept to provide a flexible mechanism for inter-module processing, control and communication. The MULTIBUS interface supports modular CPU, memory and I/O expansion in flexible, cost effective microcomputer system configurations. These configurations implement single board computers and expansion modules in a multiple processor approach to enhance system performance. This enhanced performance is achieved through partitioning of overall system functions into tasks that each of several processors can handle individually. When new system functions are added (peripherals) more processing power can be applied to handle them without impacting existing processor tasks.

Structural Features

The MULTIBUS interface is an asynchronous, multiprocessing system bus designed to perform 8-bit and 16-bit transfers between single board computers, memory and I/O expansion boards. Its interface structure consists of 24 address lines, 16 data lines, 12 control lines, 9 interrupt lines, and 6 bus exchange lines. These signal lines are implemented on single board computers and a mating

backplane in the form of two edge connectors resident on 6.75" x 12.00" form factor PC boards. The primary 86-pin P1 connector contains all MULTIBUS signal lines except the four address extension lines. The auxiliary 60-pin P2 connector contains the four MULTIBUS address extension lines, and reserves the remaining 56 pins for implementing the iLBX™ Execution Bus into the MULTIBUS system architecture.

Bus Elements

The MULTIBUS system bus supports three device categories: 1) Master, 2) Slave, 3) Intelligent Slave.

A bus master device is any module which has the ability to control the bus. This ability is not limited to only one master device. The MULTIBUS interface is capable of supporting multiple masters on the same system through bus exchange logic. Once access has been acquired by a master device, it has a period of exclusive control to affect data transfers through a generation of command signals, address signals and memory or I/O addresses.

A bus slave device is a module that decodes the address lines on the MULTIBUS and acts upon the command signals from the bus masters. Slave devices are not capable of controlling the MULTIBUS interface.

The intelligent slave has the same bus interface attributes as the slave device but also incorporates an on-

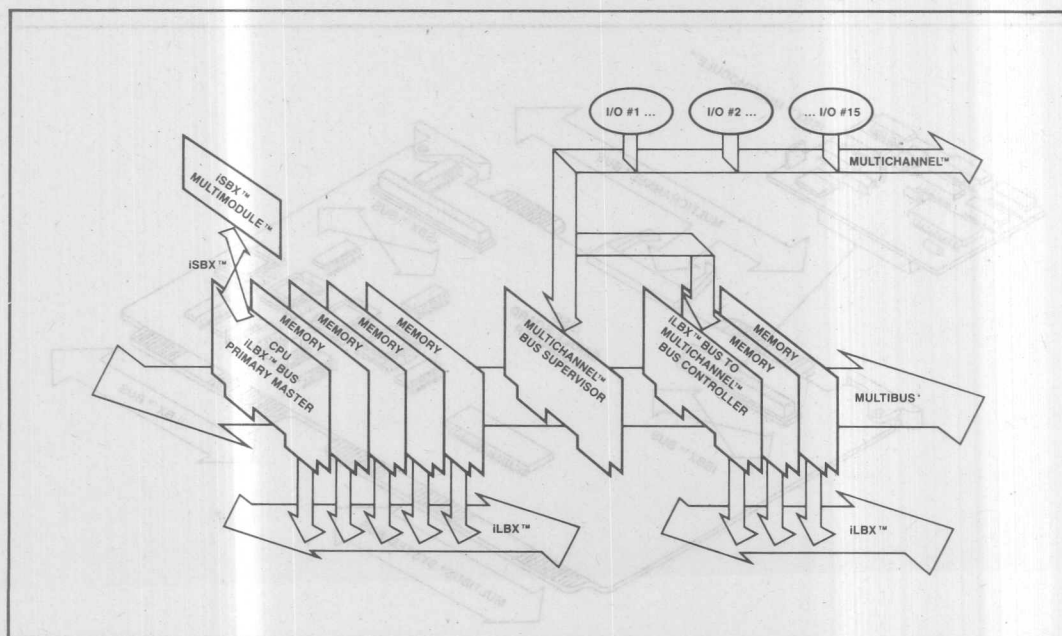


Figure 1. MULTIBUS® System Architecture

board microprocessor to control on-board memory and I/O tasks. This combination of on-board processor, memory and I/O allow the intelligent slave to complete on-board operations without MULTIBUS access.

Bus Interface/Signal Line Descriptions

The MULTIBUS system bus signal lines are grouped into five classes based on the functions they perform: 1) control lines, 2) address and inhibit lines, 3) data lines, 4) interrupt lines, 5) bus exchange lines. Figure 2 shows the implementation of these signal lines.

The MULTIBUS control lines are broken down into five sub-groups: clock signals (2), commands (4), acknowledge (1), initialize (1), and lock (1). The two clock signals provide for the generation of a master clock for the system and the synchronization of bus arbitration logic. The four command lines are the communications links between the bus masters and bus slaves, specifying types of operations to be performed such as reads or writes from memory or I/O. The transfer acknowledge line is the slave's acknowledgement that a requested

action of the master is complete. The initialize signal is generated to reset the entire system to a known state. The lock signal is used by an active bus master to lock dual-ported for mutual exclusion.

The address and inhibit lines are made up of 24 address lines, two inhibit lines, and one byte control line. The 24 address lines are signal lines used to carry the address of the memory location or the I/O device that is being referenced. These 24 lines allow a maximum of 16 million bytes of memory to be accessed. When addressing an I/O device, sixteen address lines are used to address a maximum of 64 thousand devices. The two inhibit lines are used to allow different types of memory (RAM, ROM, etc.) having the same memory address to be accessed in a preferred priority arrangement. The byte control line is used to select the upper byte of a 16-bit word in systems incorporating 16-bit memory and I/O modules.

The MULTIBUS interface supports sixteen bi-directional data lines to transmit or receive information to or from a memory location or an I/O port.

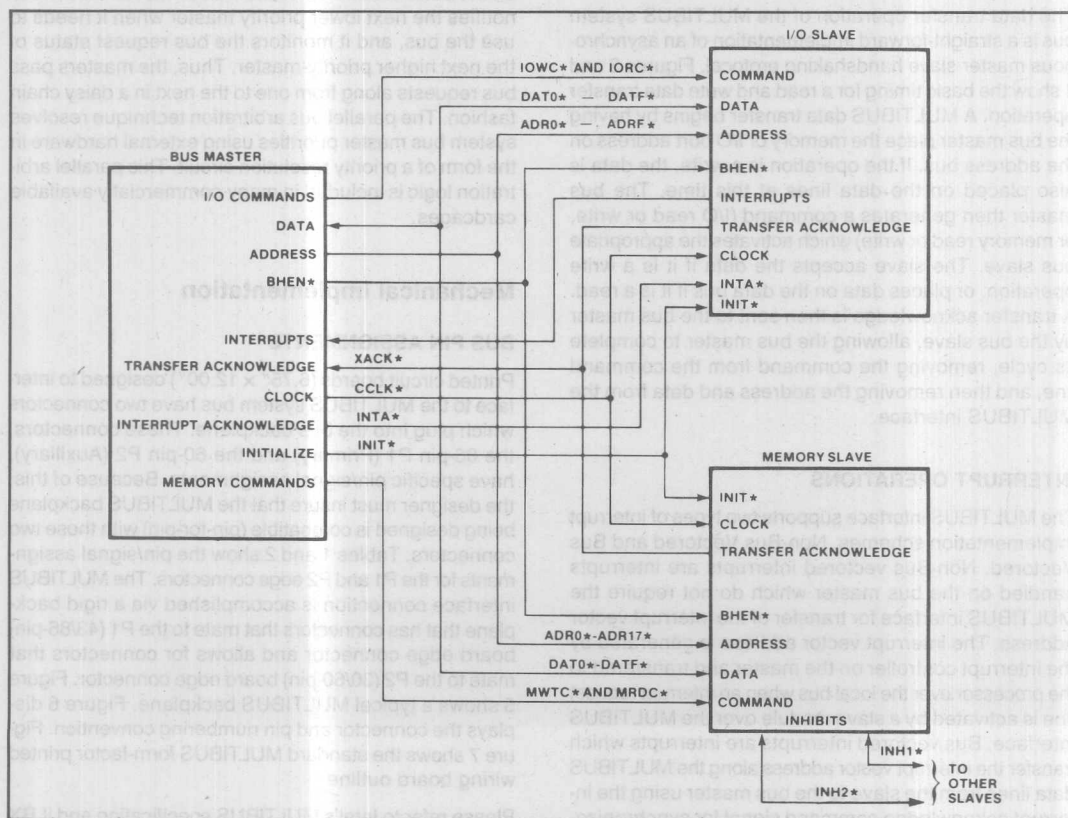


Figure 2. MULTIBUS® Interface Signal Lines

The MULTIBUS interrupt lines consist of eight interrupt request lines and one interrupt acknowledge line. Interrupts are requested by activating one of the eight interrupt request lines. The interrupt acknowledge signal is generated by the bus master when an interrupt request is received. It effectively freezes interrupt status and requests the placement of the interrupt vector address onto the data lines. There are six bus exchange lines that support two bus arbitration schemes on the MULTIBUS system bus. A bus master gains control of the bus through the manipulation of these signals. The bus request, bus priority, bus busy, and bus clock signals provide for a slot dependent priority scheme to resolve bus master contention on the MULTIBUS interface. Use of the common bus request signal line can save arbitration time by providing for a higher priority path to gain control of the system bus.

Bus Operation Protocol

DATA TRANSFER OPERATION

The data transfer operation of the MULTIBUS system bus is a straight-forward implementation of an asynchronous master-slave handshaking protocol. Figures 3 and 4 show the basic timing for a read and write data transfer operation. A MULTIBUS data transfer begins by having the bus master place the memory or I/O port address on the address bus. If the operation is a write, the data is also placed on the data lines at this time. The bus master then generates a command (I/O read or write, or memory read or write) which activates the appropriate bus slave. The slave accepts the data if it is a write operation, or places data on the data bus if it is a read. A transfer acknowledge is then sent to the bus master by the bus slave, allowing the bus master to complete its cycle, removing the command from the command line, and then removing the address and data from the MULTIBUS interface.

INTERRUPT OPERATIONS

The MULTIBUS interface supports two types of interrupt implementation schemes, Non-Bus Vectored and Bus Vectored. Non-Bus vectored interrupts are interrupts handled on the bus master which do not require the MULTIBUS interface for transfer of the interrupt vector address. The interrupt vector address is generated by the interrupt controller on the master and transferred to the processor over the local bus when an interrupt request line is activated by a slave module over the MULTIBUS interface. Bus vectored interrupts are interrupts which transfer the interrupt vector address along the MULTIBUS data lines from the slave to the bus master using the interrupt acknowledge command signal for synchronization. When an interrupt request occurs, the interrupt

control logic on the bus master interrupts the processor, generating an interrupt acknowledge command that freezes the interrupt logic on the bus for priority resolution and locks the MULTIBUS system bus. After the bus master selects the highest priority active interrupt request lines, a set of interrupt sequences allow the bus slave to put its interrupt vector address on the data lines. This address is used as a pointer to interrupt the service routine.

BUS EXCHANGE TECHNIQUES

The MULTIBUS system bus can accommodate several bus masters on the same system, each one taking control of the bus as it needs to affect data transfers. The bus masters request bus control through a bus exchange sequence.

The MULTIBUS interface provides for two bus exchange priority techniques: a serial technique and a parallel technique. In a serially arbitrated MULTIBUS system, requests for system bus access are ordered by priority on the basis of bus slot location. Each master on the bus notifies the next lower priority master when it needs to use the bus, and it monitors the bus request status of the next higher priority-master. Thus, the masters pass bus requests along from one to the next in a daisy chain fashion. The parallel bus arbitration technique resolves system bus master priorities using external hardware in the form of a priority resolution circuit. This parallel arbitration logic is included in many commercially available cardcages.

Mechanical Implementation

BUS PIN ASSIGNMENTS

Printed circuit boards (6.75" x 12.00") designed to interface to the MULTIBUS system bus have two connectors which plug into the bus backplane. These connectors, the 86-pin P1 (Primary) and the 60-pin P2 (Auxiliary), have specific pin/signal assignments. Because of this, the designer must insure that the MULTIBUS backplane being designed is compatible (pin-for-pin) with these two connectors. Tables 1 and 2 show the pin/signal assignments for the P1 and P2 edge connectors. The MULTIBUS interface connection is accomplished via a rigid backplane that has connectors that mate to the P1 (43/86-pin) board edge connector and allows for connectors that mate to the P2 (30/60-pin) board edge connector. Figure 5 shows a typical MULTIBUS backplane. Figure 6 displays the connector and pin numbering convention. Figure 7 shows the standard MULTIBUS form-factor printed wiring board outline.

Please refer to Intel's MULTIBUS specification and iLBX bus specification for more detailed information.

Table 1. MULTIBUS® Pin/Signal Assignment — (P1)

	Pin	(Component Side)		Pin	(Circuit Side)	
		Mnemonic	Description		Mnemonic	Description
Power Supplies	1	GND	Signal GND	2	GND	Signal GND
	3	+5V	+5Vdc	4	+5V	+5Vdc
	5	+5V	+5Vdc	6	+5V	+5Vdc
	7	+12V	+12Vdc	8	+12V	+12Vdc
	9		Reserved, bussed	10		Reserved, bussed
	11	GND	Signal GND	12	GND	Signal GND
Bus Controls	13	BCLK *	Bus Clock	14	INIT *	Initialize
	15	BPRN *	Bus Pri. In	16	BPRO *	Bus Pri. Out
	17	BUSY *	Bus Busy	18	BREQ *	Bus Request
	19	MRDC *	Mem Read Cmd	20	MWTC *	Mem Write Cmd
	21	IORC *	I/O Read Cmd	22	IOWC *	I/O Write Cmd
	23	XACK *	XFER Acknowledge	24	INH1 *	Inhibit 1 (disable RAM)
Bus Controls and Address	25	LOCK *	Lock	26	INH2 *	Inhibit 2 (disable PROM or ROM)
	27	BHEN *	Byte High Enable	28	AD10 *	Address Bus
	29	CBRQ *	Common Bus Request	30	AD11 *	
	31	CCLK *	Constant Clk	32	AD12 *	
	33	INTA *	Intr Acknowledge	34	AD13 *	
Interrupts	35	INT6 *	Parallel Interrupt	36	INT7 *	Parallel Interrupt
	37	INT4 *	Requests	38	INT5 *	Requests
	39	INT2 *		40	INT3 *	
	41	INT0 *		42	INT1 *	
Address	43	ADRE *	Address Bus	44	ADRF *	Address Bus
	45	ADRC *		46	ADRD *	
	47	ADRA *		48	ADRB *	
	49	ADR8 *		50	ADR9 *	
	51	ADR6 *		52	ADR7 *	
	53	ADR4 *		54	ADR5 *	
	55	ADR2 *		56	ADR3 *	
	57	ADR0 *		58	ADR1 *	
Data	59	DATE *	Data Bus	60	DATF *	Data Bus
	61	DATC *		62	DATD *	
	63	DATA *		64	DATB *	
	65	DAT8 *		66	DAT9 *	
	67	DAT6 *		68	DAT7 *	
	69	DAT4 *		70	DAT5 *	
	71	DAT2 *		72	DAT3 *	
	73	DAT0 *		74	DAT1 *	
Power Supplies	75	GND	Signal GND	76	GND	Signal GND
	77		Reserved, bussed	78		Reserved, bussed
	79	-12V	-12Vdc	80	-12V	-12Vdc
	81	+5V	+5Vdc	82	+5V	+5Vdc
	83	+5V	+5Vdc	84	+5V	+5Vdc
	85	GND	Signal GND	86	GND	Signal GND

All Reserved pins are reserved for future use and should not be used if upwards compatibility is desired.

*Note: The Reserved MULTIBUS P2 connector pin/signal assignments are contained in Intel's iLBX Bus Specification.

Table 2. MULTIBUS® Pin/Signal Assignment — (P2)

	Pin	(Component Side)		Pin	(Circuit Side)	
		Mnemonic	Description		Mnemonic	Description
	1		Reserved	2		Reserved
	3		Reserved	4		Reserved
	5		Reserved	6		Reserved
	7		Reserved	8		Reserved
	9		Reserved	10		Reserved
	11		Reserved	12		Reserved
	13		Reserved	14		Reserved
	15		Reserved	16		Reserved
	17		Reserved	18		Reserved
	19		Reserved	20		Reserved
	21		Reserved	22		Reserved
	23		Reserved	24		Reserved
	25		Reserved	26		Reserved
	27		Reserved	28		Reserved
	29		Reserved	30		Reserved
	31		Reserved	32		Reserved
	33		Reserved	34		Reserved
	35		Reserved	36		Reserved
	37		Reserved	38		Reserved
	39		Reserved	40		Reserved
	41		Reserved	42		Reserved
	43		Reserved	44		Reserved
	45		Reserved	46		Reserved
	47		Reserved	48		Reserved
	49		Reserved	50		Reserved
	51		Reserved	52		Reserved
	53		Reserved	54		Reserved
Address	55	ADR16*	Address Bus	56	ADR17*	Address Bus
	57	ADR14*		58	ADR15*	
	59		Reserved, Bussed	60		Reserved, Bussed

All Reserved Pins are reserved for future use and should not be used if upwards compatibility is desired.

*Note: The Reserved MULTIBUS P2 connector pin/signal assignments are contained in Intel's iLBX Bus Specification.

SPECIFICATION

Word Size

Data — 8 and 16-bit

Memory Addressing

24-bits — 16 megabyte — direct access

I/O Addressing

16-bits — 64 Kbytes

Maximum Bus Backplane Length

18 inches

Bus Devices Supported

16 total devices — (Master, Slave, Intelligent Slave)

Bus Bandwidth

10 megabytes/sec — 16-bit

5 megabytes/sec — 8-bit

Bus Exchange Cycle

200 nsec — Best Case; 300 nsec — Worst Case (assuming no bus master is currently active on the bus.)

Electrical Characteristics

BUS POWER SUPPLY SPECIFICATIONS

Table 3.

Parameter	Standard ¹			
	Ground	+5	+12	-12
Mnemonic	GND	+5V	+12V	-12V
Bus Pins	P1-1,2,11,12, 75,76,85,86	P1-3,4,5,6, 81,82,83, 84	P1-7,8	P1-79,80
Tolerance	Ref.	±1%	±1%	±1%
Combined Line & Load Reg	Ref.	0.1%	0.1%	0.1%
Ripple (Peak to Peak)	Ref.	50 mV	50 mV	50 mV
Transient Response (50% Load Change)		100 μ s	100 μ s	100 μ s

¹Point of measurement is at connection point between motherboard and power supply. At any card edge connector a degradation of 2% maximum (e.g. voltage tolerance $\pm 2\%$) is allowed.

BUS TIMING

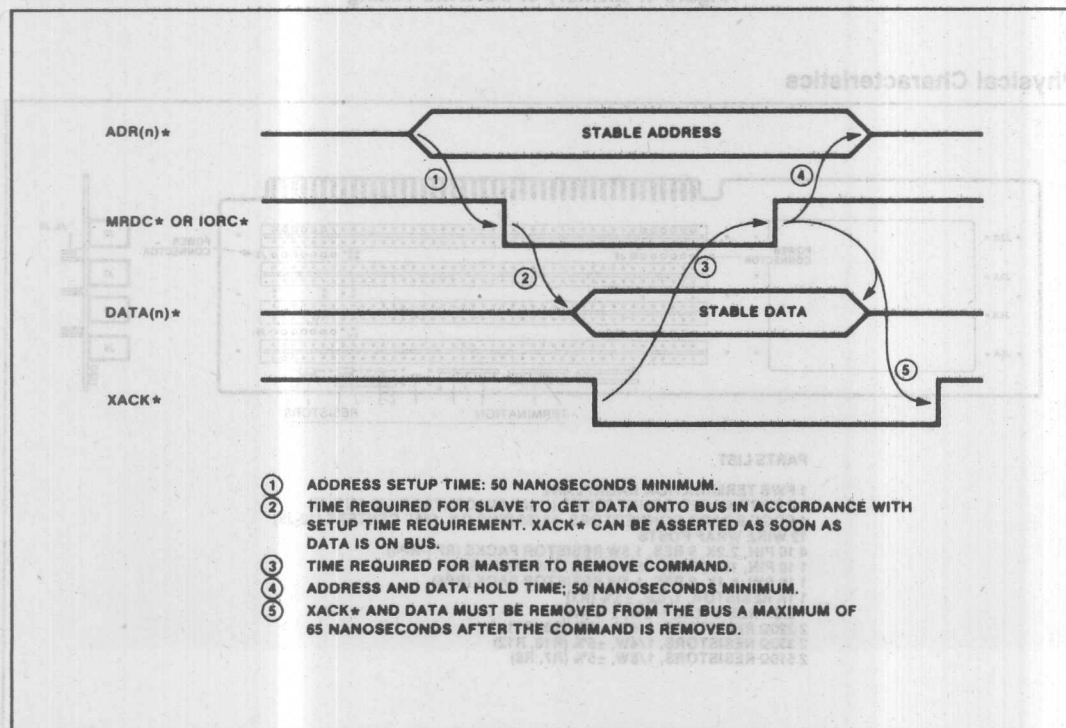


Figure 3. Memory or I/O Read Timing

BUS TIMING (Con't)

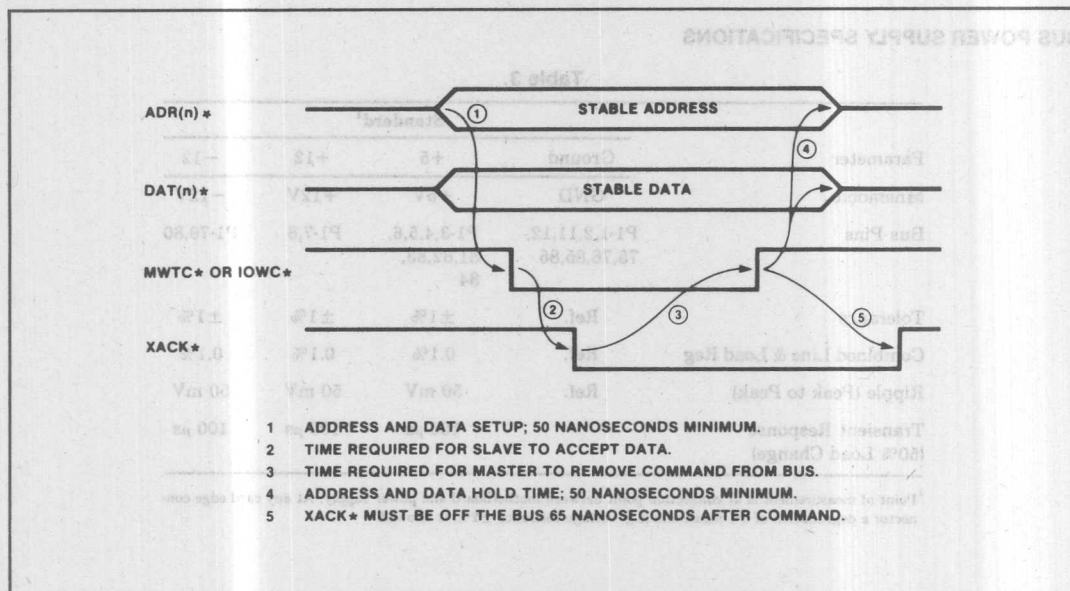


Figure 4. Memory or I/O Write Timing

Physical Characteristics

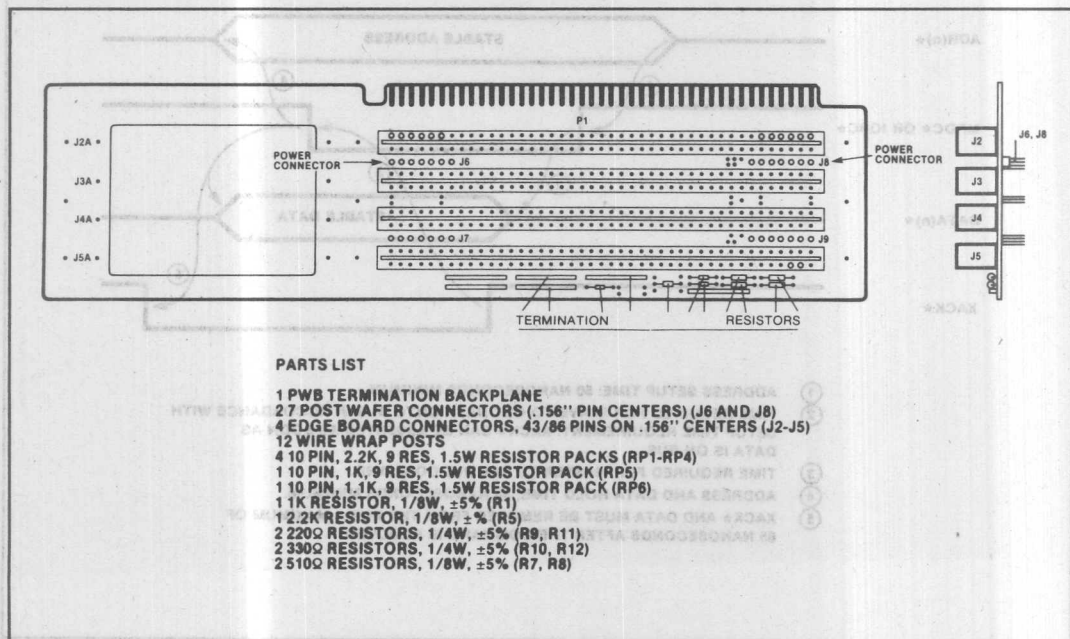


Figure 5. MULTIBUS® System Backplane Example

PHYSICAL CHARACTERISTICS (Con't)

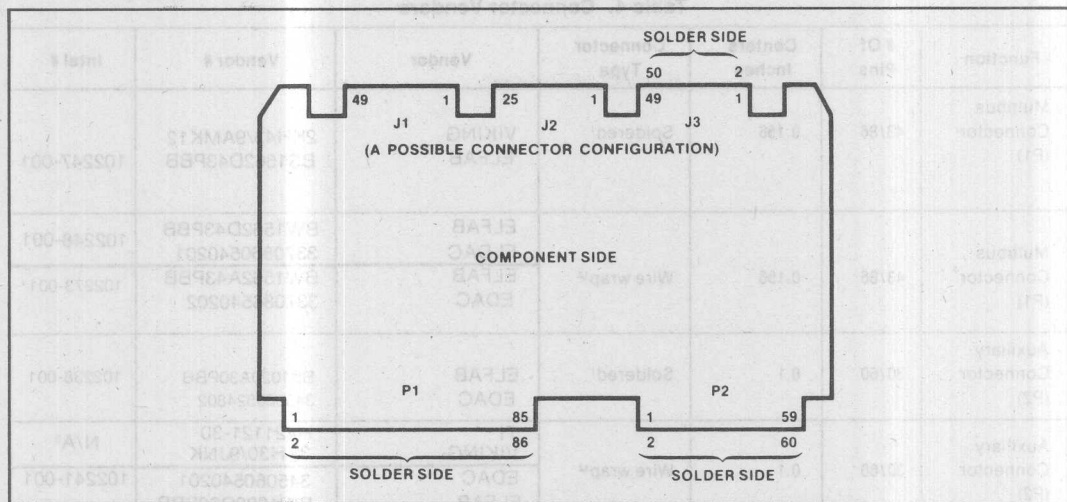


Figure 6. Connector and Pin Numbering

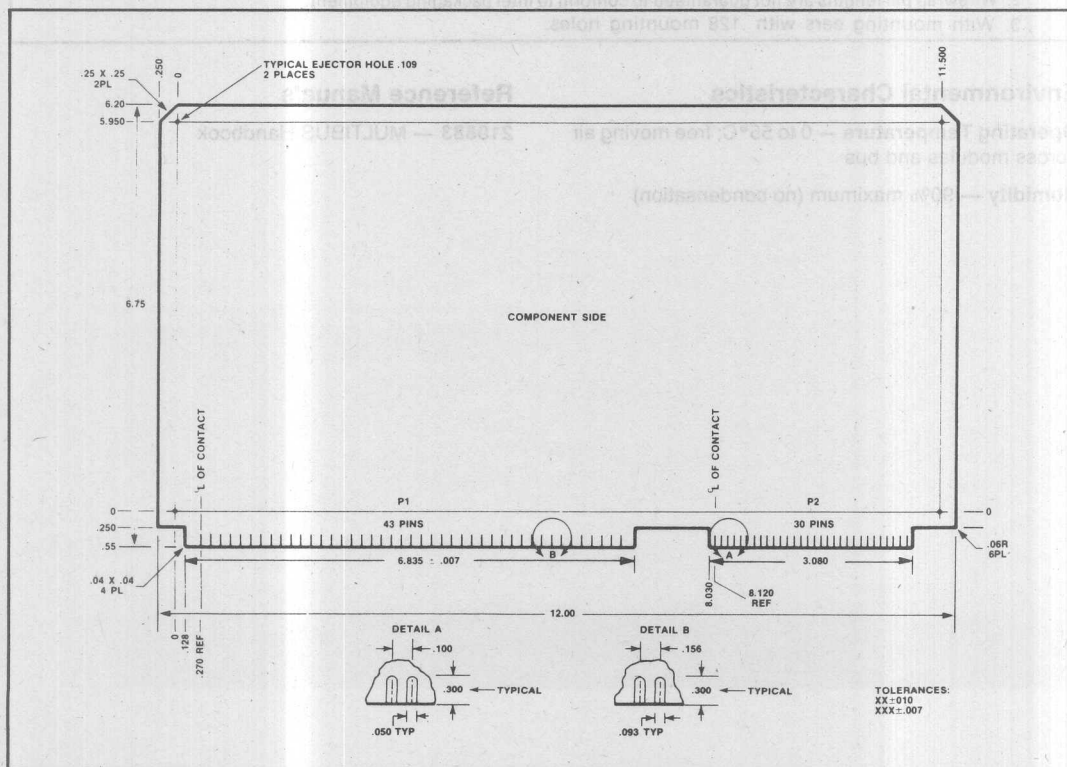


Figure 7. Standard Printed Wiring Board Outline

Backplane Connectors

Table 4. Connector Vendors

Function	# Of Pins	Centers Inches	Connector Type	Vendor	Vendor #	Intel #
Multibus Connector (P1)	43/86	0.156	Soldered'	VIKING ELFAB	2KH43/9AMK12 BS1562D43PBB	102247-001
Multibus Connector (P1)	43/86	0.156	Wire wrap ²	ELFAB ELDAC	BW1562D43PBB 3370860540201	102248-001
				ELFAB EDAC	BW1562A43PBB 337086540202	102273-001
Auxiliary Connector (P2)	30/60	0.1	Soldered'	ELFAB EDAC	BS1020A30PBB 345060524802	102238-001
Auxiliary Connector (P2)	30/60	0.1	Wire wrap ²	TI VIKING	H421121-30 3KH30/9JNK	N/A ³
				EDAC ELFAB	345060540201 BW1020D30PBB	102241-001
Notes:						
1. Connector heights are not guaranteed to conform to Intel packaging equipment.						
2. Wirewrap pin lengths are not guaranteed to conform to Intel packaging equipment.						
3. With mounting ears with .128 mounting holes.						

Environmental Characteristics

Reference Manuals

Operating Temperature — 0 to 55°C; free moving air across modules and bus

210883 — MULTIBUS Handbook

Humidity — 90% maximum (no condensation)

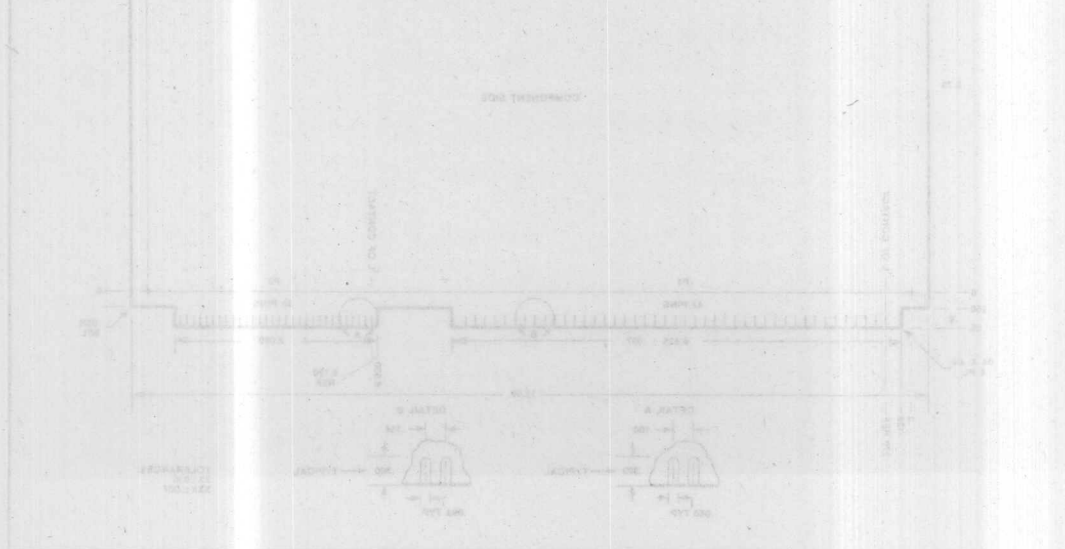
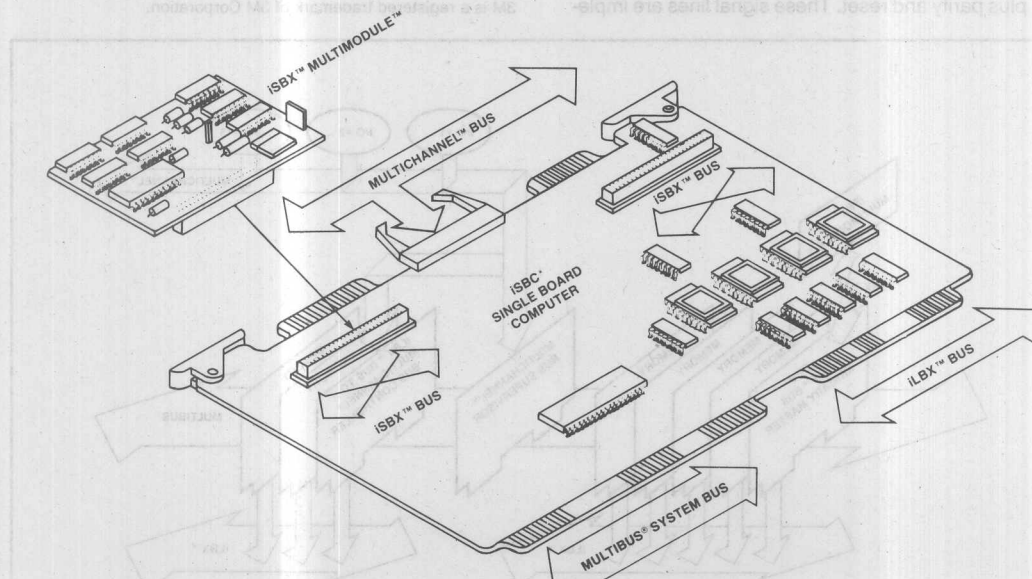


Figure 7. Standard Printed Wiring Board Outline

MULTICHANNEL™ I/O BUS

- High speed 8- or 16-bit block transfers between memory and/or I/O
- Transfer rates up to 8 megabytes/sec.
- Full speed operation at distances of up to 15 meters.
- Supports Supervisor, Controller, or basic Talker/Listener capabilities
- Off-loads burst mode I/O activities from host CPU and MULTIBUS® system bus
- Up to 16 devices may be interfaced to the bus.
- 16 megabytes of memory and 16 megabytes of I/O are addressable on each device

The MULTICHANNEL™ I/O Bus is one of a family of standard bus structures resident within Intel's total system architecture. The MULTICHANNEL bus is a general purpose, high-speed I/O bus capable of significantly increasing system performance by providing a separate data path for DMA I/O activities. By isolating I/O transfers from the system bus, the MULTICHANNEL bus off-loads I/O activity from the host CPU, reduces the probability of bus saturation on the system bus, and reduces contention between I/O and data processing activities on the system bus. The MULTICHANNEL bus can support up to 16 devices at distances up to 15 meters with a maximum burst throughput of 8 megabytes per second. These 16 devices are classified in a manner similar to the IEEE 488 bus concept: Supervisors, Controllers, or Talker and Listeners. As a non-proprietary, standardized I/O bus, the MULTICHANNEL bus is a cost-effective DMA interface ideal for applications such as computer graphics, specialized peripheral control, automatic test equipment, video camera image processing, data acquisition, and high-speed MULTIBUS® system-to-system communication.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, Multichannel and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Architectural Overview

The MULTICHANNEL bus is the standard high speed I/O interface to MULTIBUS-based systems. Its general purpose design and high performance (8 MB/sec) augment the overall system design by improving I/O interface flexibility and system throughput. The flexibility is realized by using an easy-to-use public standard interface that can support up to sixteen 8-bit or 16-bit devices at up to 15 meters. This structure allows the MULTICHANNEL bus to provide easy I/O system expansion, effective box-to-box communication, and a growth path capable of supporting new generations of high-performance I/O devices. The MULTICHANNEL bus increases system throughput by providing a high-performance data path for efficient movement of large amounts of data.

Structural Features

MULTICHANNEL™ BUS CONFIGURATION

The MULTICHANNEL bus is a multiplexed, asynchronous block transfer, 16-bit I/O bus designed to handle 8-bit and 16-bit transfers between peripherals and single board computers. Its structure (pictured in Figure 2) consists of 16 address/data lines, 6 control lines, 2 interrupt lines, plus parity and reset. These signal lines are imple-

mented as either a 60 conductor flat ribbon cable or a twisted-pair cable spanning a distance of up to 15 meters. A 30/60-pin 3M® connector is recommended for device connection to the MULTICHANNEL bus. The male connectors are installed on each MULTICHANNEL device and the female connectors are mounted on the cable. To insure system integrity, the MULTICHANNEL cable is terminated at both ends.

BUS ELEMENTS

Three device types — the Basic device, the bus Controller device, and the bus Supervisor device — each provide a different level of capability. The Basic Talker/Listener device has lowest capability, responding only to data transfer requests issued by a Supervisor or Controller. The bus Controller device has higher capability than a Basic Talker/Listener on the bus. It can respond to data transfer requests, control data transfers, and can program other MULTICHANNEL devices under direction from a bus Supervisor. Operating at the highest capability is the bus Supervisor device. It provides major control and management of the MULTICHANNEL bus. The bus Supervisor resolves and grants MULTICHANNEL bus priority, monitors bus status, handles interrupts, and controls the reset line, in addition to performing all bus Controller functions.

3M is a registered trademark of 3M Corporation.

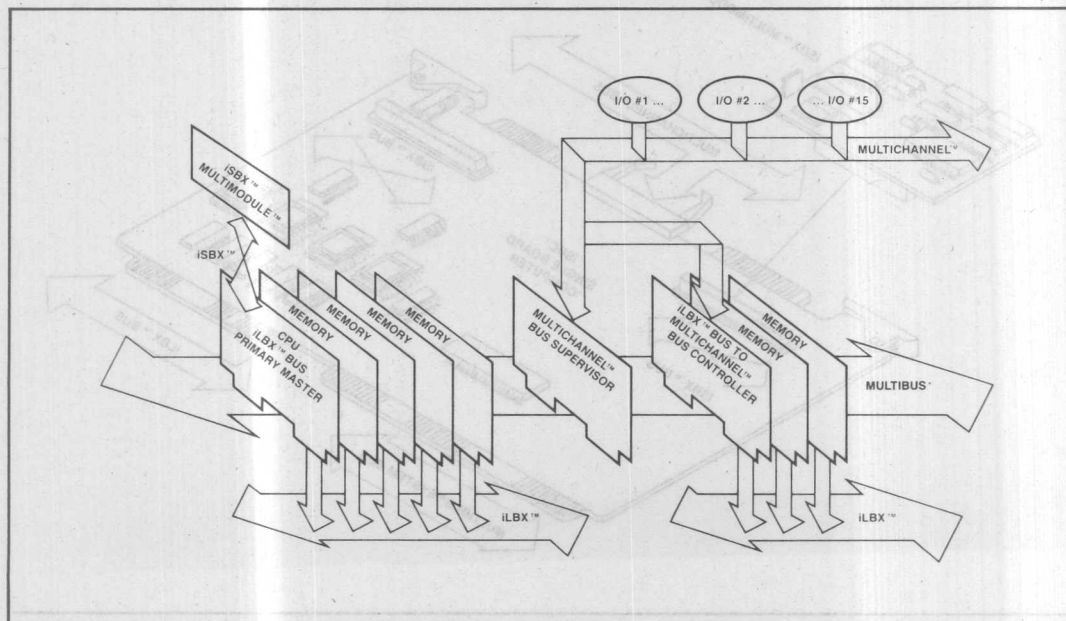


Figure 1. MULTIBUS® System Architecture

MULTICHANNEL bus devices are functionally flexible, creating overlaps between types of bus functions and types of bus devices performing those functions. These devices perform functions in various states of operation: master, slave, talker, listener. When a device is controlling the command/action lines, it is in the master state, and both the bus Supervisor and the bus Controller can operate in this state, although not simultaneously. The slave state indicates a device that can monitor the command/action lines. Only Controllers and Basic Talker/Listeners operate as slaves. All three device types can operate in the talker state or the listener state, but not all at the same time. A Talker is any device selected by the bus master which is writing data to the bus. A Listener is any selected device which is reading data from the bus.

BUS INTERFACE/SIGNAL LINE DESCRIPTIONS

The MULTICHANNEL bus signal lines are grouped into five classes based on the functions they perform: address/data, control, interrupt, parity, and reset. The 16 address/data lines are multiplexed by a control line to act either as 16 unidirectional address lines or 16 bidirectional data lines. When used as address lines, they transmit the device address to all devices attached to

the MULTICHANNEL bus. When used as bidirectional data lines, they transmit and receive data to or from MULTICHANNEL devices. The six control lines determine the overall operation of the bus from specifying the type of data transfer to providing the handshake for data transfers between MULTICHANNEL devices. Two interrupt lines are supplied to initiate and terminate data transfers, and to indicate device failures, memory failures, or parity errors. A parity line and a reset line provide support for a parity option and system reset capability whenever required.

BUS PIN ASSIGNMENTS

For proper MULTICHANNEL implementation, a 60 conductor (twisted pair or flat) cable using a 30/60 pin 3M connector, is used for device connection to the bus. Figure 3 is an outline drawing of the iSBC® MULTICHANNEL connector which also shows the pin numbering. The MULTICHANNEL bus connector signal pin assignments are listed in Table 1. Cable termination is implemented at both cable ends to insure proper system integrity over a 15-meter cable. Figure 4 is a schematic of the cable termination circuits. A cable termination module could be created that would then be connected to the cable end via a 30/60 pin connector.

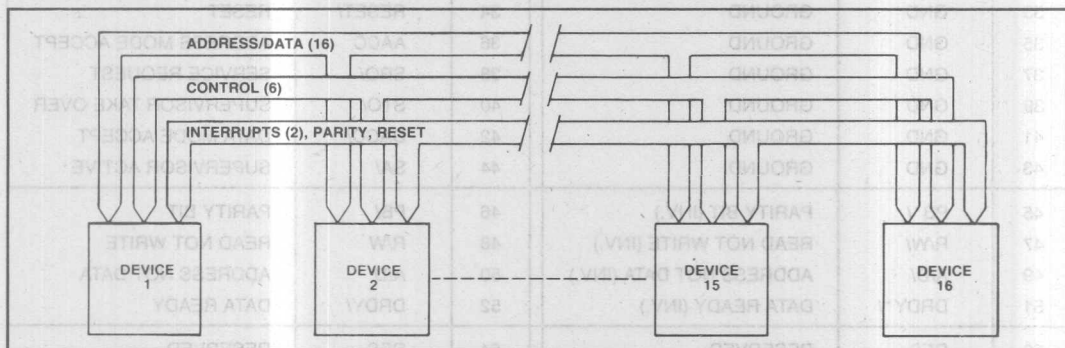


Figure 2. Block Diagram of MULTICHANNEL™ Bus Structure

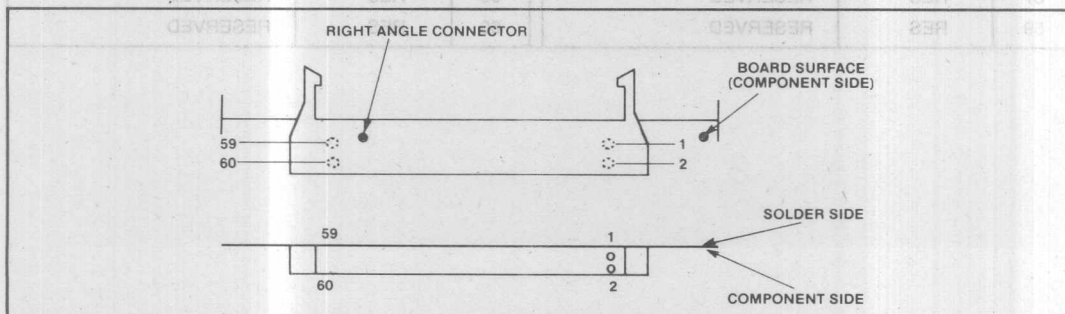
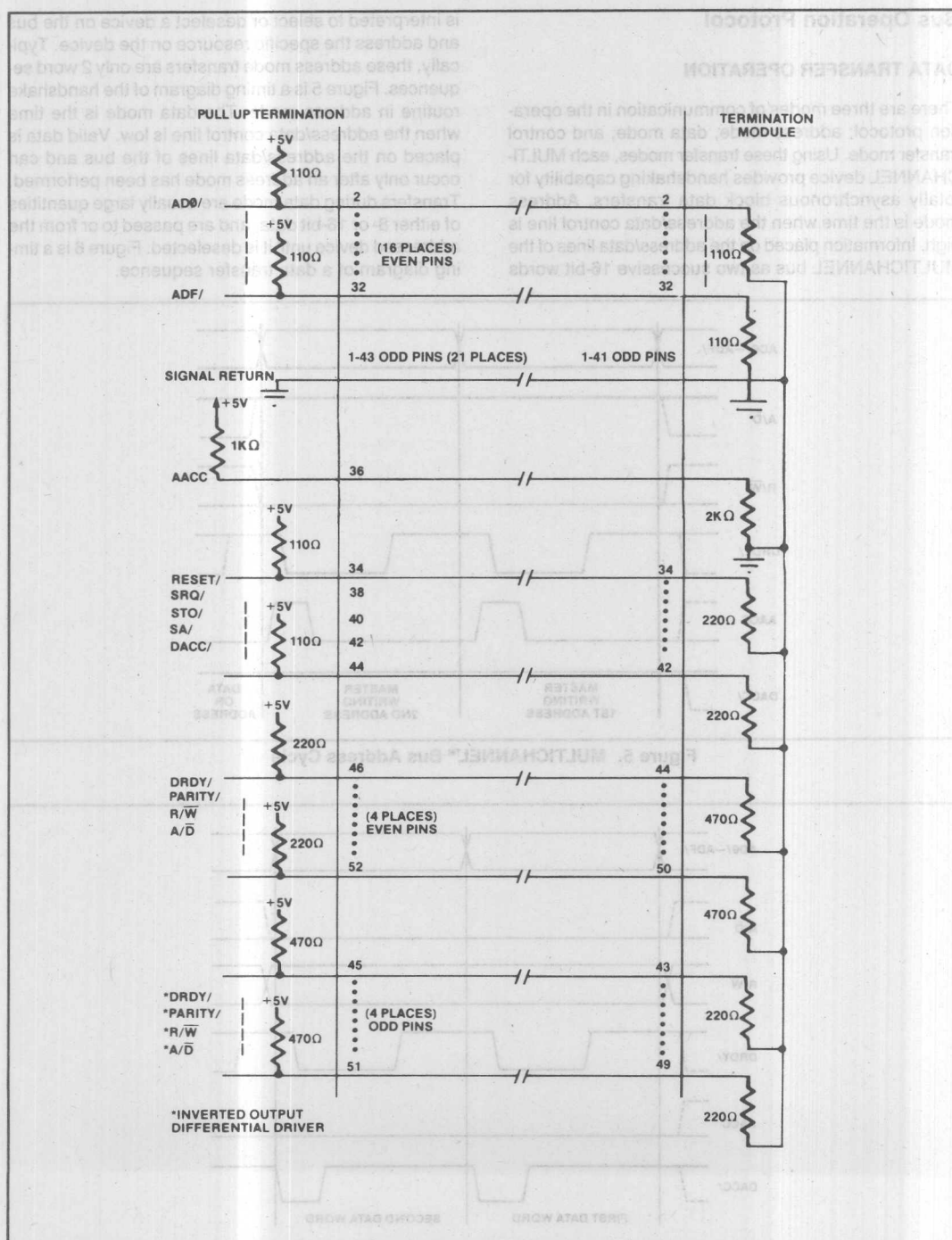


Figure 3. Connector Example

Table 1. MULTICHANNEL™ Bus Pin Assignments

Lower Row			Upper Row		
Pin	Mnemonic	Signal Name	Pin	Mnemonic	Signal Name
1	GND	GROUND	2	AD0/	ADDRESS DATA LINE 0
3	GND	GROUND	4	AD1/	ADDRESS DATA LINE 1
5	GND	GROUND	6	AD2/	ADDRESS DATA LINE 2
7	GND	GROUND	8	AD3/	ADDRESS DATA LINE 3
9	GND	GROUND	10	AD4/	ADDRESS DATA LINE 4
11	GND	GROUND	12	AD5/	ADDRESS DATA LINE 5
13	GND	GROUND	14	AD6/	ADDRESS DATA LINE 6
15	GND	GROUND	16	AD7/	ADDRESS DATA LINE 7
17	GND	GROUND	18	AD8/	ADDRESS DATA LINE 8
19	GND	GROUND	20	AD9/	ADDRESS DATA LINE 9
21	GND	GROUND	22	ADA/	ADDRESS DATA LINE 10
23	GND	GROUND	24	ADB/	ADDRESS DATA LINE 11
25	GND	GROUND	26	ADC/	ADDRESS DATA LINE 12
27	GND	GROUND	28	ADD/	ADDRESS DATA LINE 13
29	GND	GROUND	30	ADE/	ADDRESS DATA LINE 14
31	GND	GROUND	32	ADF/	ADDRESS DATA LINE 15
33	GND	GROUND	34	RESET/	RESET
35	GND	GROUND	36	AACC	ADDRESS MODE ACCEPT
37	GND	GROUND	38	SRQ/	SERVICE REQUEST
39	GND	GROUND	40	STO/	SUPERVISOR TAKE OVER
41	GND	GROUND	42	DACC/	DATA MODE ACCEPT
43	GND	GROUND	44	SA/	SUPERVISOR ACTIVE
45	PB*/	PARITY BIT (INV.)	46	PB/	PARITY BIT
47	R/W/	READ NOT WRITE (INV.)	48	R/W	READ NOT WRITE
49	A/D/	ADDRESS NOT DATA (INV.)	50	A/D	ADDRESS NOT DATA
51	DRDY*/	DATA READY (INV.)	52	DRDY/	DATA READY
53	RES	RESERVED	54	RES	RESERVED
55	RES	RESERVED	56	RES	RESERVED
57	RES	RESERVED	58	RES	RESERVED
59	RES	RESERVED	60	RES	RESERVED



Bus Operation Protocol

DATA TRANSFER OPERATION

There are three modes of communication in the operation protocol: address mode, data mode, and control transfer mode. Using these transfer modes, each MULTICHANNEL device provides handshaking capability for totally asynchronous block data transfers. Address mode is the time when the address/data control line is high. Information placed on the address/data lines of the MULTICHANNEL bus as two successive 16-bit words

is interpreted to select or deselect a device on the bus and address the specific resource on the device. Typically, these address mode transfers are only 2 word sequences. Figure 5 is a timing diagram of the handshake routine in address mode. The data mode is the time when the address/data control line is low. Valid data is placed on the address/data lines of the bus and can occur only after an address mode has been performed. Transfers during data mode are usually large quantities of either 8- or 16-bit data, and are passed to or from the addressed device until it is deselected. Figure 6 is a timing diagram of a data transfer sequence.

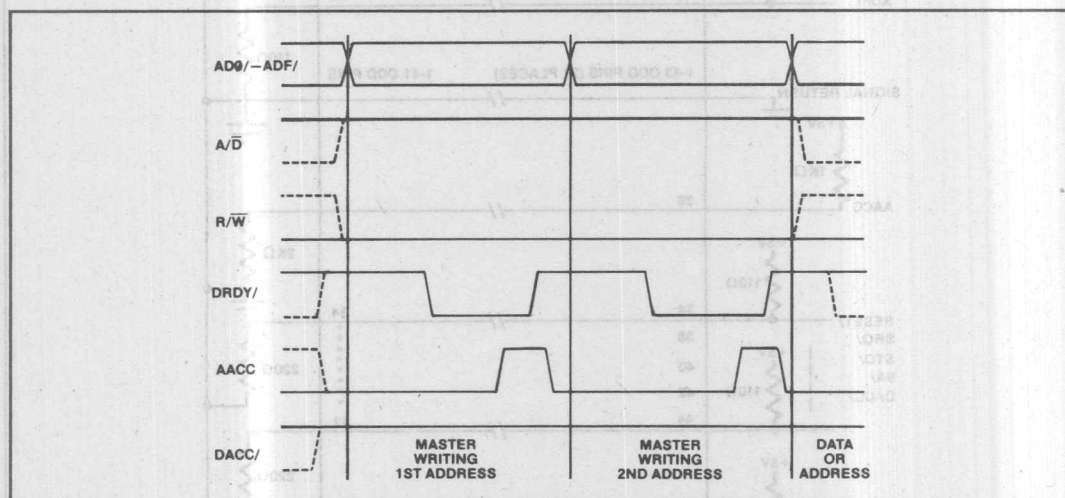


Figure 5. MULTICHANNEL™ Bus Address Cycle

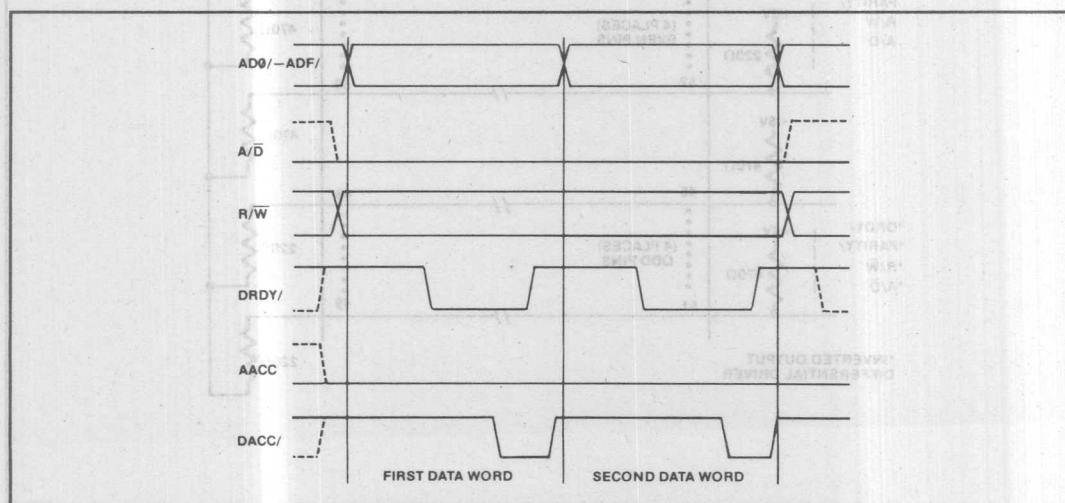


Figure 6. MULTICHANNEL™ Bus Data Transfer Sequence

Control transfer mode is the time when the bus Supervisor selects the bus Controller and programs its registers with required information. Once programmed, a bus Controller may select a device and originate a data transfer operation.

The operational sequences of these transfer modes are similar in handling read and write operations to and from the 16 megabytes of memory and the 16 megabytes of registers addressable on each MULTICHANNEL device.

A typical transfer sequence begins when the master sends a two-word address sequence to select a MULTICHANNEL device and specify address, direction and resource (memory vs. I/O) of the data transfer. Following device selection, the Talker proceeds to send the data as a continuous 8 or 16-bit data word stream until the block data move is complete. The master terminates the transfer by issuing another two-word address sequence for device deselection.

The transfer sequence described is identical for both memory and register type transfers. The master controls similar read and write operations between devices, and the address select and deselect sequences use the same address format. Figure 7 contains the MULTICHANNEL bus address format.

DEVICE REGISTER DEFINITION

Of the 16 megabytes of register space per device, the first 16 registers are pre-defined to provide a standard register area common to all devices. The remaining registers are user definable. Table 2 lists the 16 defined registers along with their function. The use of this register concept allows for standard interface between all MULTICHANNEL devices. Please refer to the MULTICHANNEL Bus Specification for more detailed information.

Table 2. MULTICHANNEL™ Device Register Definitions

Register Number	Definition	Mode
0	STO/ Flag/Status	Read Only
1	SRQ/ Flag/Status	Read Only
2	SRQ/ Mask	Write Only
3	Device Command	Write Only
4	Device Parameter	Write Only
5	Data Address 1	Read or Write
6	Data Address 2	Read or Write
7	Block Length 1	Read or Write
8	Block Length 2	Read or Write
9	Error Address 1	Read Only
10	Error Address 2	Read Only
11	Address Extension	Write Only
12-15	Reserved	
16-16 Mbyte	User Defined	Read or Write

BUS INTERRUPT HANDLING

The MULTICHANNEL bus Supervisor, being responsible for bus access and control, monitors the two bus interrupt lines. The Supervisor Take-Over interrupt (STO) is used to inform the bus Supervisor that a device wants to return control of the bus to the Supervisor or that an error has occurred. The Service Request Interrupt (SRQ) is used by devices which do not have control of the bus, but require service from the bus Supervisor. To locate a device transmitting a bus interrupt, the bus Supervisor

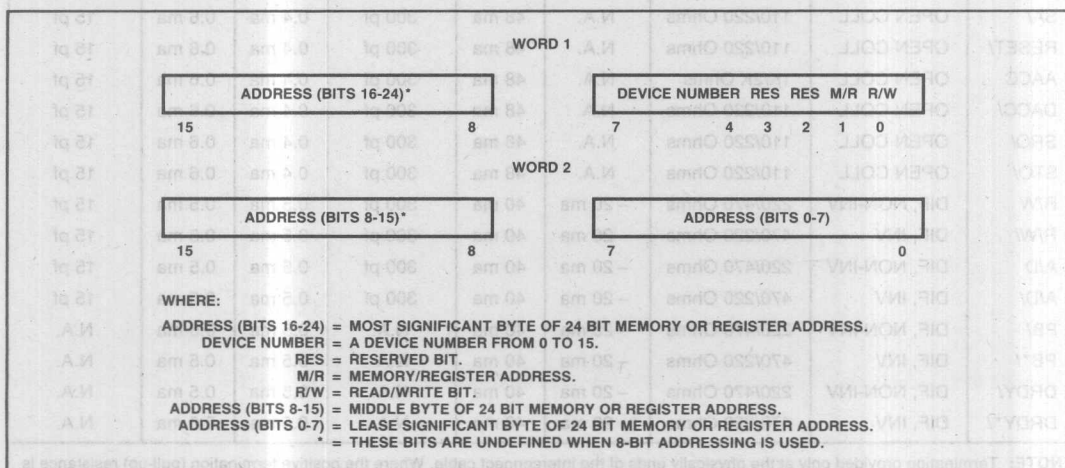


Figure 7. MULTICHANNEL™ Bus Address Format

polls each device attached to the bus by reading the appropriate register of each device and testing for a non-zero value. In current implementations, the Supervisor polls each device only once. If the interrupt is not removed an error occurs.

PARITY AND RESET

Parity operation on the MULTICHANNEL bus is provided, but is not required. The bus Supervisor selects

between parity mode and non-parity mode depending upon system requirements. If parity mode is selected all Talkers must generate odd parity. All active Listeners monitor the parity line and generate an STO interrupt signal if there is a parity error.

A reset function is also supported by the MULTICHANNEL bus, and is controlled by the bus Supervisor to bring the bus to a known state. It is used to reset all devices after power-up, and when required to gain control of the bus.

SPECIFICATIONS

Word Size

Data — 8, 16-bit

Memory Addressing

24-bits — 16 megabyte — direct access — automatic incrementing

Register Addressing

24-bits — 16 megabyte — direct access

Electrical Characteristics

DC SPECIFICATIONS

Maximum Bus Length

15 meters (50 feet)

Bus Devices Supported

16 total devices — (Supervisor, Controller, and Talker/Listener)

Bus Bandwidth

8 megabytes/sec. — 16-bit

4 megabytes/sec. — 8-bit

Table 3: DC Specifications

Signal Name	Driver Type	Termination (see Note)	Min. Driver Requirements			Max. Receiver Requirements		
			High	Low	Load Cap	High	Low	Load Cap
AD15-0/	TRI-STATE	110 Ohms	— 5 ma	48 ma	300 pf	0.2 ma	0.8 ma	15 pf
SA/	OPEN COLL	110/220 Ohms	N.A.	48 ma	300 pf	0.4 ma	0.6 ma	15 pf
RESET/	OPEN COLL	110/220 Ohms	N.A.	48 ma	300 pf	0.4 ma	0.6 ma	15 pf
AACC	OPEN COLL	1K/2K Ohms	N.A.	48 ma	300 pf	0.4 ma	0.6 ma	15 pf
DACC/	OPEN COLL	110/220 Ohms	N.A.	48 ma	300 pf	0.4 ma	0.6 ma	15 pf
SRQ/	OPEN COLL	110/220 Ohms	N.A.	48 ma	300 pf	0.4 ma	0.6 ma	15 pf
STO/	OPEN COLL	110/220 Ohms	N.A.	48 ma	300 pf	0.4 ma	0.6 ma	15 pf
R/W	DIF, NON-INV	220/470 Ohms	—20 ma	40 ma	300 pf	0.5 ma	0.5 ma	15 pf
R/W/	DIF, INV	470/220 Ohms	—20 ma	40 ma	300 pf	0.5 ma	0.5 ma	15 pf
A/D	DIF, NON-INV	220/470 Ohms	—20 ma	40 ma	300 pf	0.5 ma	0.5 ma	15 pf
A/D/	DIF, INV	470/220 Ohms	—20 ma	40 ma	300 pf	0.5 ma	0.5 ma	15 pf
PB/	DIF, NON-INV	220/470 Ohms	—20 ma	40 ma	N.A.	0.5 ma	0.5 ma	N.A.
PB*/	DIF, INV	470/220 Ohms	—20 ma	40 ma	N.A.	0.5 ma	0.5 ma	N.A.
DRDY/	DIF, NON-INV	220/470 Ohms	—20 ma	40 ma	N.A.	0.5 ma	0.5 ma	N.A.
DRDY*/	DIF, INV	470/220 Ohms	—20 ma	40 ma	N.A.	0.5 ma	0.5 ma	N.A.

NOTE: Termination provided only at the physically ends of the interconnect cable. Where the positive termination (pull-up) resistance is different from the negative termination (pull-down) resistance, the positive termination resistance is listed first.

BUS TIMING

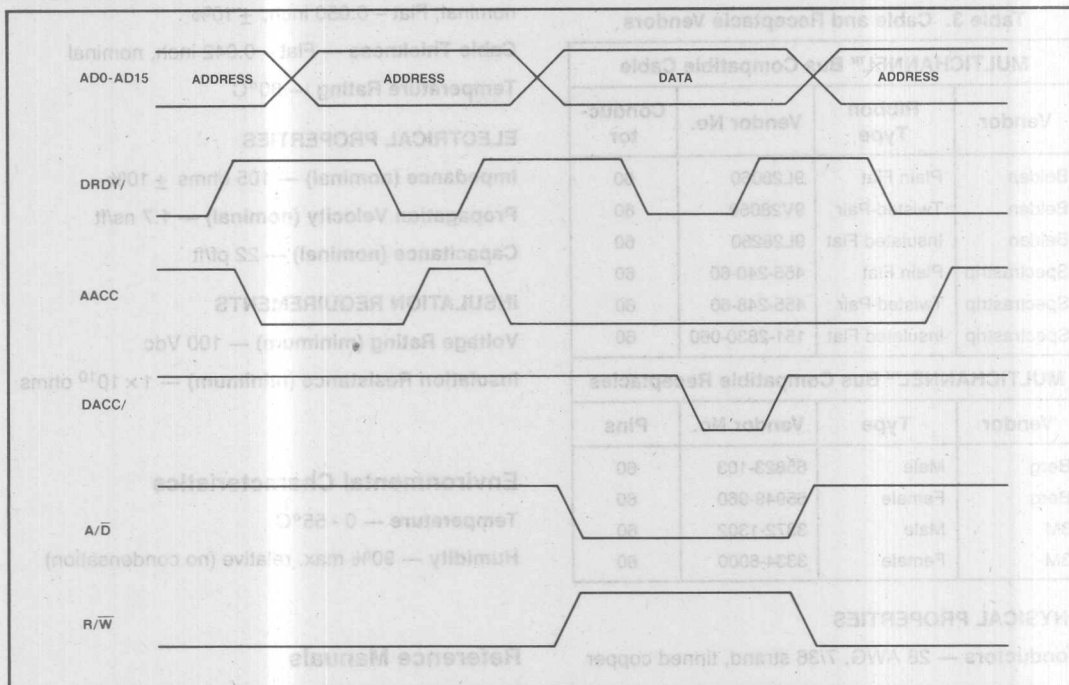


Figure 8. Address-Read-Address-Write Cycles

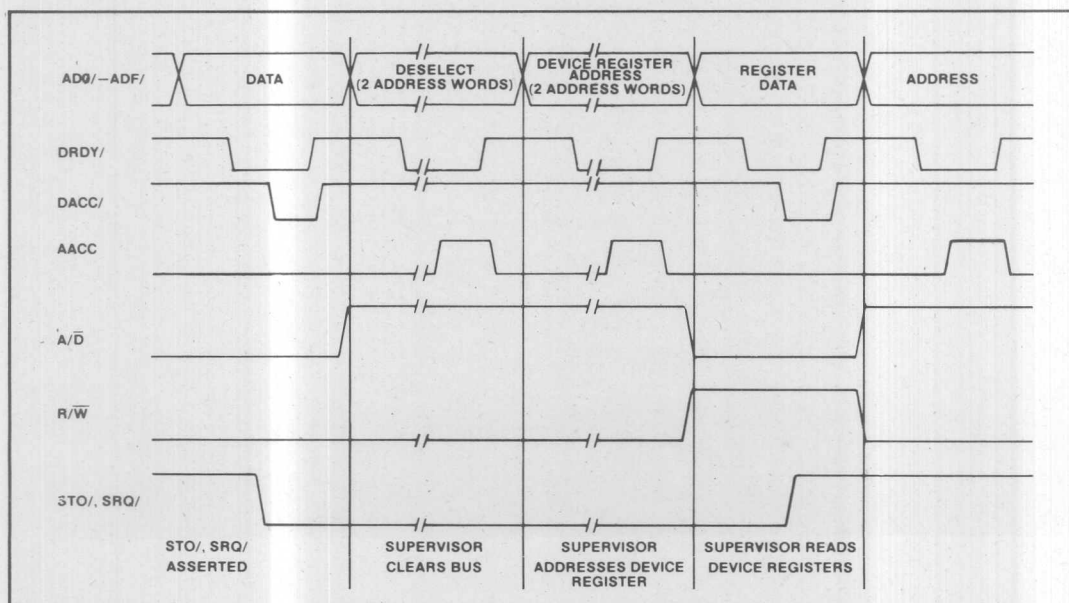


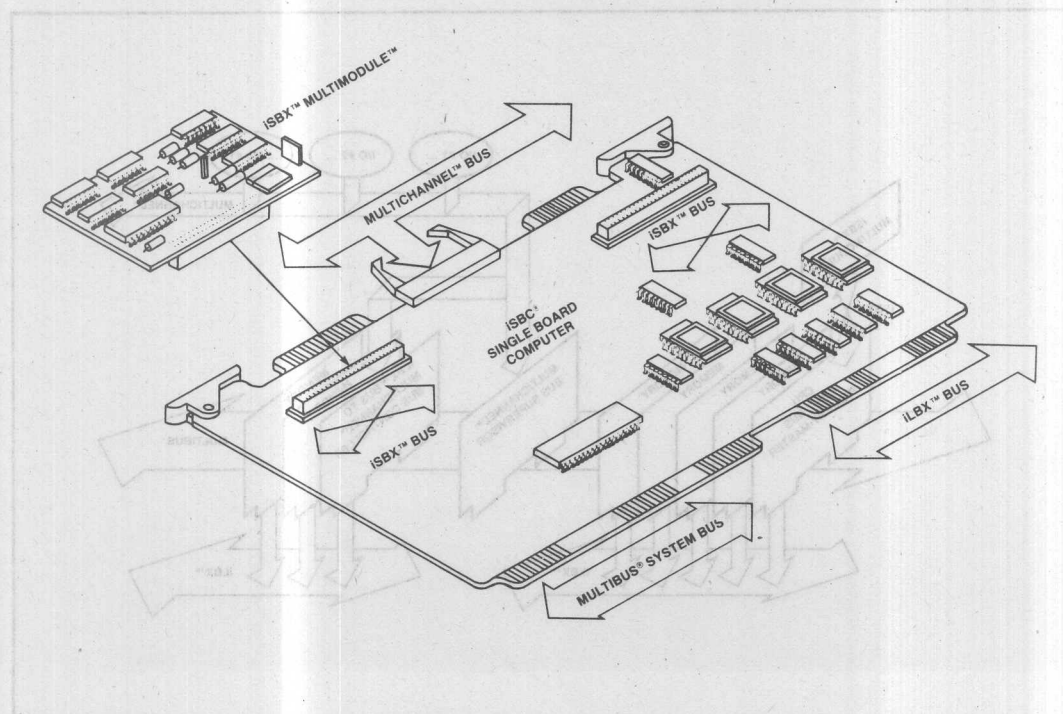
Figure 9. Supervisor Interrupt Timing

iLBX™ EXECUTION BUS

- **High bus bandwidth**
 - 9.5 Mbytes/sec. for 8-bit transfers
 - 19 Mbytes/sec. for 16-bit transfers
- **16 Mbyte addressing range**
- **8 and 16-bit data transfers**

- **Supports up to 5 iLBX™ compatible devices per bus**
- **Primary and secondary master bus exchange capabilities**
- **Standard 60-pin MULTIBUS® P2 connector**

The iLBX™ Execution Bus is one of a family of standard bus structures resident within Intel's total system architecture. The Local Bus Extension (iLBX) Bus is a dedicated execution bus capable of significantly increasing system performance by extending the processor board's on-board local bus to off-board resources. This extension provides for arbitration-free, direct access to high-performance memory. Acting as a "virtual" iSBC®, up to 16 megabytes of processor addressable memory can be accessed over the iLBX bus and appear as though it were resident on the processor board. The iLBX Bus preserves advantages in performance and architecture of on-board memory, while allowing memory configurations larger than possible on a single board computer. High throughput and independence from MULTIBUS® activities make the iLBX bus an ideal solution for "working store" type program memory and data processing applications requiring large amounts of high performance memory. Such applications include graphics systems, robotics, process control, office systems, and CAD/CAM.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, IMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, Multichannel and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supercedes previously published specifications on these devices from Intel.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Architectural Overview

The iLBX bus is an architectural solution for supporting large amounts of high performance memory. It is the first structure that allows the CPU board selection to be decoupled from the on-board memory requirement, and still maximizes the processor's performance potential. It eliminates the processor's need to access its off-board memory resources solely over the MULTIBUS system bus. Architectural consistency with the single board computer approach including iLBX memory can be maintained by dual port access of memory resources between the iLBX bus and the MULTIBUS system bus. This allows for global access by other processors and I/O devices while still providing high speed local CPU operations. This sub-system created by the iLBX bus of a single board computer and a maximum of 4 memory cards can be perceived architecturally as a "virtual single board computer". The implementation of iLBX bus "virtual modules" makes it possible to create functional modules with a new level of flexibility and performance in implementing a wide range of memory capabilities. With future needs in mind, the iLBX bus has the capability of accessing a full 16 megabytes of memory.

Structural Features

The iLBX bus uses a non-multiplexed 16-bit configuration capable of 8 and 16-bit transfers. Used in conjunction with the MULTIBUS interface, the iLBX bus resides on the MULTIBUS form factor P2 connector and supercedes the MULTIBUS interface definitions for the P2 signals. The iLBX bus uses the standard 60-pin MULTIBUS P2 connector and occupies 56 of the P2 connector pins with 16 data lines, 24 address lines plus control, command access, and parity signals. The four MULTIBUS address extension lines on the MULTIBUS/iLBX P2 connector retain the standard MULTIBUS interface definition.

Bus Elements

The iLBX bus supports three distinct device categories: 1) Primary Master, 2) Secondary Master, 3) Slave. These three device types may be combined to create several iLBX local busses ranging (in size) from a minimum of two to a maximum of five devices per iLBX bus. There is only one Primary Master in any given implementation of iLBX bus, and its presence is required along with the attachment of at least one Slave device. To provide alternate access over an iLBX bus, one optional Sec-

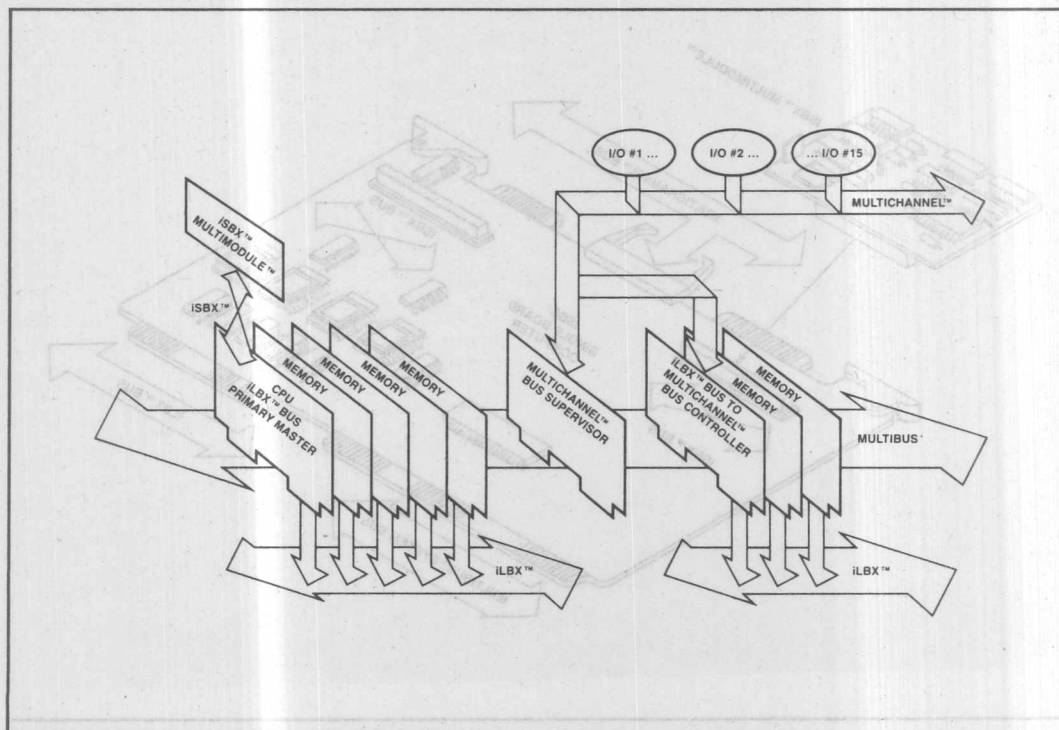


Figure 1. MULTIBUS® System Architecture

ondary Master may be incorporated to create a "two-master" local bus subsystem. By limiting the iLBX bus to two masters (a Primary and a Secondary), bus arbitration is reduced to a simple request and acknowledge process, with privileged use of the bus maintained by the Primary Master, and limited access granted to the Secondary Master when needed.

The Primary Master executes the role of iLBX bus "supervisor" by controlling the general operation of the bus and managing Secondary Master accesses to the Slave memory resources.

The Secondary Master Device is an option providing alternate access to the Slave resources on the iLBX bus. Secondary master devices are typically DMA driven. This feature is provided for implementation flexibility when occasional DMA transfers in and out of iLBX memory resources can optimize the overall system performance. The Secondary Master essentially duplicates the Primary Master's data transfer capability, but must rely on the Primary Master to grant access. Once access is granted, the Secondary Master controls the bus, and drives all signal lines until the operation is complete and control is passed back to the Primary Master.

The Slave devices contain the memory resources used by the Primary Master and the optional Secondary Master. Each iLBX implementation can contain a maximum of four Slave devices. Using 64K RAM technology on four slave devices with ECC can provide for over 2 megabytes of "on-board" high performance memory. With 256K RAM chips, each iLBX bus could contain slave devices with memory totalling 8 megabytes. As memory technology increases, the iLBX bus is designed to incorporate it in rapid fashion because it is capable of directly accessing a full 16 megabytes of memory on its high-performance Slave devices.

Bus Interface/Signal Line Descriptions

The iLBX bus interface is divided into four functional classes of signal lines: address and data lines, control lines, command lines, and bus access lines. The 40 address and data lines defined by the iLBX Bus Specification consist of 16 data lines and 24 address lines.

There are 16 bi-directional data lines exclusively used to handle 8-bit and 16-bit data transfers between the active bus master and the selected slave device. The iLBX bus uses these data lines for all data transfers, and are driven by tri-state drivers.

The 24 address lines on the iLBX bus provide the ability to directly address 16 megabytes of memory. These single-direction address lines are exclusively driven by the active bus master. The iLBX bus master uses them to select a specific slave device. Three control lines

specify the type of data transfer between master and slave devices, while the three command lines initiate, control, and terminate the transfer. There are also three bus access lines used to transfer bus control between master devices.

Bus Pin Assignments

The iLBX bus uses the standard 60-pin MULTIBUS P2 connector. The physical location of each pin assignment and its corresponding function is listed in Table 1. The four MULTIBUS address extension lines (pins 55-58 on the P2 connector) retain the standard MULTIBUS interface functions.

Bus Operation Protocol

The operation protocol for the iLBX bus is a straightforward set of procedures consisting of three basic operations: bus control access, write data to memory, read data from memory. These operations use asynchronous protocol with positive acknowledgment.

Bus Access

The iLBX bus is shared by at most two masters; one Primary Master and one optional Secondary Master, each providing an alternate access path to iLBX bus memory resources. The mechanism for obtaining bus access is a simple request and acknowledge process communicated between masters. Each master is a bus controller of similar capabilities, responsible for data transfer operations between devices, but the Primary Master has the added responsibility of controlling iLBX bus accesses.

The Primary Master has default control of the iLBX bus. If the Secondary Master needs access to the bus, it must initiate a request and wait for acknowledgment from the Primary Master. The choice of when to surrender control of the bus rests with the Primary Master, but if no data transfer is in progress, the Primary Master normally relinquishes control immediately to the Secondary Master.

Data Transfer Operation

The iLBX bus supports two types of data transfer operations: write data to memory and read data from memory. These data transfer operations facilitate the passing of information between the active bus master and the selected slave device. The operation of these two transfer types is very similar; the only differences being the direction of the data transfer and the device driving the data lines.

For either type of data transfer, the active bus master first initiates the transfer operation by placing the memory address on the address lines (AB23-AB0) and a con-

trol configuration on the control lines to select the slave device. Once the slave device is selected, the type of data transfer becomes the key factor. With the write operation, the active master maintains control of the data lines and provides valid data within the specified time. Upon accepting a data element, the slave sends a receipt acknowledgment signal to the master which completes the data transfer operation.

With the read operation, the slave device drives the data lines and places valid data on the data lines before sampling by the active master. The slave acknowledges the master to signal the end of the data transfer, and the master completes the operation.

The iLBX Bus Specification includes provisions for both optimized and non-optimized data transfers. Optimized

Table 1. iLBX™ Bus Pin Assignments, P2 Edge Connector

Component Side			Solder Side		
16-Bit Pin	Mnemonic	Signal Name	16-Bit Pin	Mnemonic	Signal Name
1	DB0	DATA LINE 0	2	DB1	DATA LINE 1
3	DB2	DATA LINE 2	4	DB3	DATA LINE 3
5	DB4	DATA LINE 4	6	DB5	DATA LINE 5
7	DB6	DATA LINE 6	8	DB7	DATA LINE 7
9	GND	GROUND	10	DB8	DATA LINE 8
11	DB9	DATA LINE 9	12	DB10	DATA LINE 10
13	DB11	DATA LINE 11	14	DB12	DATA LINE 12
15	DB13	DATA LINE 13	16	DB14	DATA LINE 14
17	DB15	DATA LINE 15	18	GND	GROUND
19	AB0	ADDRESS LINE 0	20	AB1	ADDRESS LINE 1
21	AB2	ADDRESS LINE 2	22	AB3	ADDRESS LINE 3
23	AB4	ADDRESS LINE 4	24	AB5	ADDRESS LINE 5
25	AB6	ADDRESS LINE 6	26	AB7	ADDRESS LINE 7
27	GND	GROUND	28	AB8	ADDRESS LINE 8
29	AB9	ADDRESS LINE 9	30	AB10	ADDRESS LINE 10
31	AB11	ADDRESS LINE 11	32	AB12	ADDRESS LINE 12
33	AB13	ADDRESS LINE 13	34	AB14	ADDRESS LINE 14
35	AB15	ADDRESS LINE 15	36	GND	GROUND
37	AB16	ADDRESS LINE 16	38	AB17	ADDRESS LINE 17
39	AB18	ADDRESS LINE 18	40	AB19	ADDRESS LINE 19
41	AB20	ADDRESS LINE 20	42	AB21	ADDRESS LINE 21
43	AB22	ADDRESS LINE 22	44	AB23	ADDRESS LINE 23
45	GND	GROUND	46	ACK*	SLAVE ACKNOWLEDGE
47	BHEN	BYTE HIGH ENABLE	48	R/W	READ NOT WRITE
49	ASTB*	ADDRESS STROBE	50	DSTB*	DATA STROBE
51	SMRQ*	SECONDARY MASTER REQUEST	52	SMACK*	SECONDARY MASTER ACKNOWLEDGE
53	LOCK*	ACCESS LOCK	54	GND	GROUND
55	ADR22*	MULTIBUS® ADDRESS EXTENSION LINE 22	56	ADR23*	MULTIBUS® ADDRESS EXTENSION LINE 23
57	ADR20*	MULTIBUS® ADDRESS EXTENSION LINE 20	58	ADR21*	MULTIBUS® ADDRESS EXTENSION LINE 21
59	RES	RESERVED	60	TPAR*	TRANSFER PARITY

operation uses pipelining and signal overlapping techniques to manage the data transfer timing relationships between the active bus master and the selected slave. The use of signal overlapping requires that every device attached to the iLBX bus provide a means for varying the timing of the slave request and acknowledge signals. The non-optimized operation uses fixed signal sequences, instead of signal overlapping, to assure a valid data transfer, and a device does not need a variable request or acknowledge to read data-valid timing on the iLBX bus. Please refer to the iLBX Bus Specification for detailed descriptions of these transfer operations.

Mechanical Implementation

Because the iLBX bus uses the P2 connector of the MULTIBUS form factor, the iLBX bus "shares" a MULTIBUS chassis with the MULTIBUS backplane system bus in the system design. The iLBX mechanical specifications are synonymous with the MULTIBUS specifications for board-to-board spacing, board thickness, component lead length, and component height above the board. The iLBX bus interconnection can use either flexible ribbon cable or a rigid backplane. The iLBX bus

interconnect maximum length is limited to 10 cm (approximately 4 inches); that is sufficient to span 5 card slots across two connected chassis. Figure 2 shows an iLBX bus cable assembly.

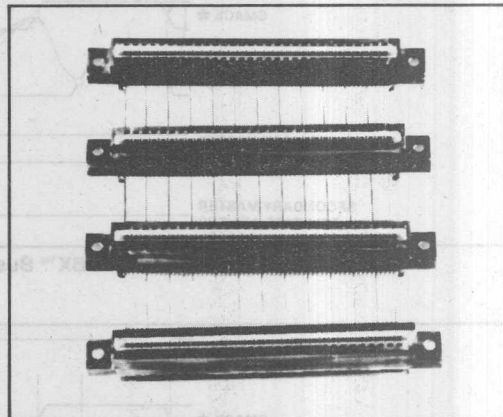


Figure 2. Typical iLBX™ Bus Interface Cable Assembly

SPECIFICATIONS

Word Size

Data — 8 and 16-bit

Memory Addressing

24-bits — 16 megabyte — direct access

Bus Bandwidth

9.5 megabytes/sec — 8-bit

19 megabytes/sec — 16-bit

Electrical Characteristics

DC SPECIFICATIONS

Table 2. DC Specifications

Signal Name	Driver Type	Termination (to +5 Vdc) At Master	Min. Driver Requirements			Max. Receiver Requirements		
			High	Low	Load Cap.	High	Low	Load Cap.
DB15-0	TRI-STATE	10K Ohms	0.4 ma	9 ma	75 pf	0.15 ma	2 ma	18 pf
TPAR*	TRI-STATE	10K Ohms	0.4 ma	9 ma	75 pf	0.15 ma	2 ma	18 pf
AB23-0	TRI-STATE	None	0.4 ma	20 ma	120 pf	0.10 ma	5 ma	30 pf
R/W	TRI-STATE	None	0.2 ma	8 ma	75 pf	0.05 ma	2 ma	18 pf
BHEN	TRI-STATE	None	0.2 ma	8 ma	75 pf	0.05 ma	2 ma	18 pf
LOCK*	TRI-STATE	None	0.2 ma	8 ma	75 pf	0.05 ma	2 ma	18 pf
SMRQ*	TTL	10K Ohms	0.05 ma	8 ma	20 pf	0.05 ma	2 ma	18 pf
SMACK*	TTL	None	0.05 ma	2 ma	20 pf	0.05 ma	2 ma	18 pf
†ASTB*	TRI-STATE	10K Ohms	0.2 ma	9 ma	75 pf	0.05 ma	2 ma	18 pf
†DSTB*	TRI-STATE	10K Ohms	0.2 ma	9 ma	75 pf	0.05 ma	2 ma	18 pf
ACK*	OPEN COLL.	330 Ohms	N.A.	20 ma	45 pf	0.05 ma	2 ma	18 pf

† At slave, series RC termination to GND (100 ohm, 10 pf)

BUS TIMING

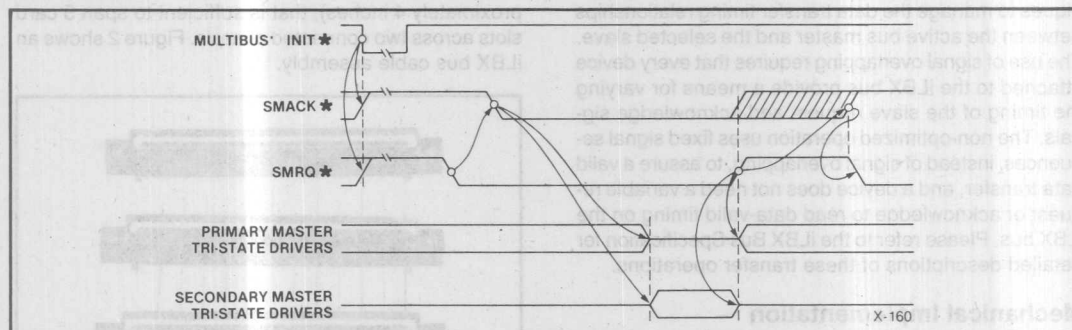


Figure 3. iLBX™ Bus Granting Timing Chart

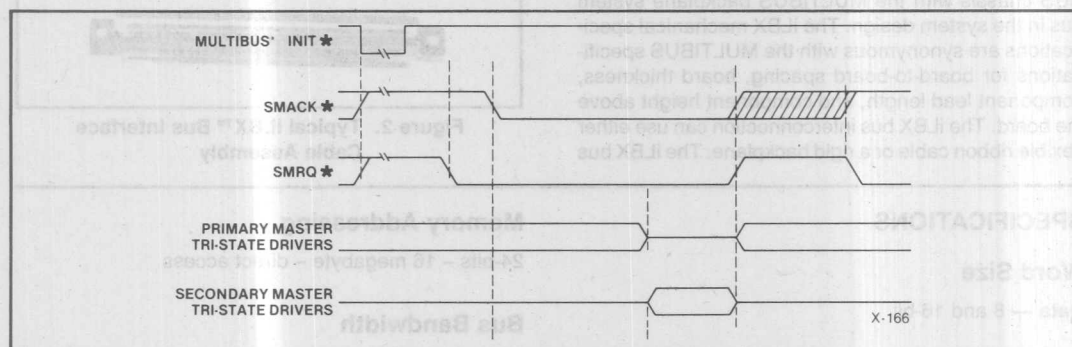


Figure 4. iLBX™ Bus Control Transfer Timing Chart

16-Bit Transfer Timing —

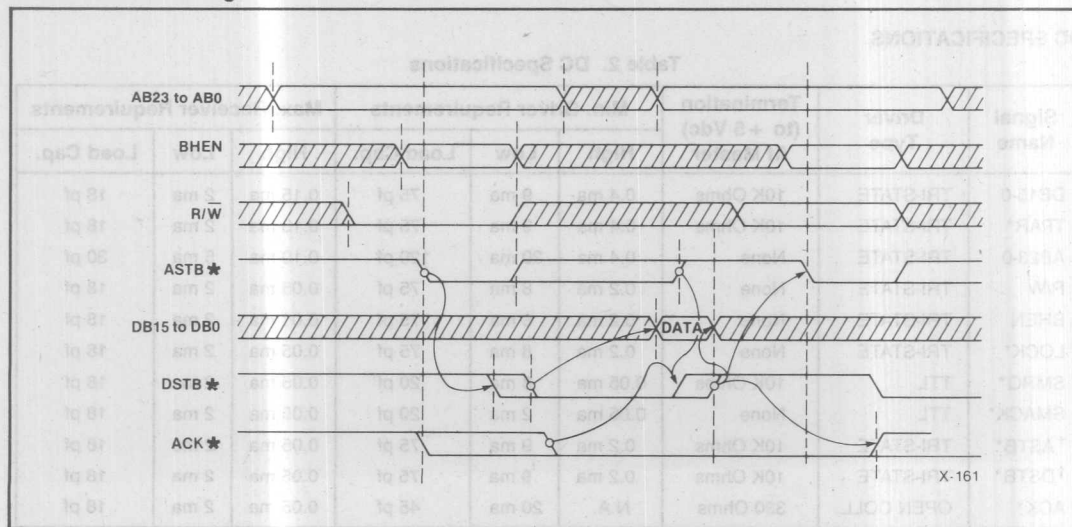


Figure 5. Write Data-To-Memory

BUS TIMING

16-Bit Transfer Timing (Con't.) —

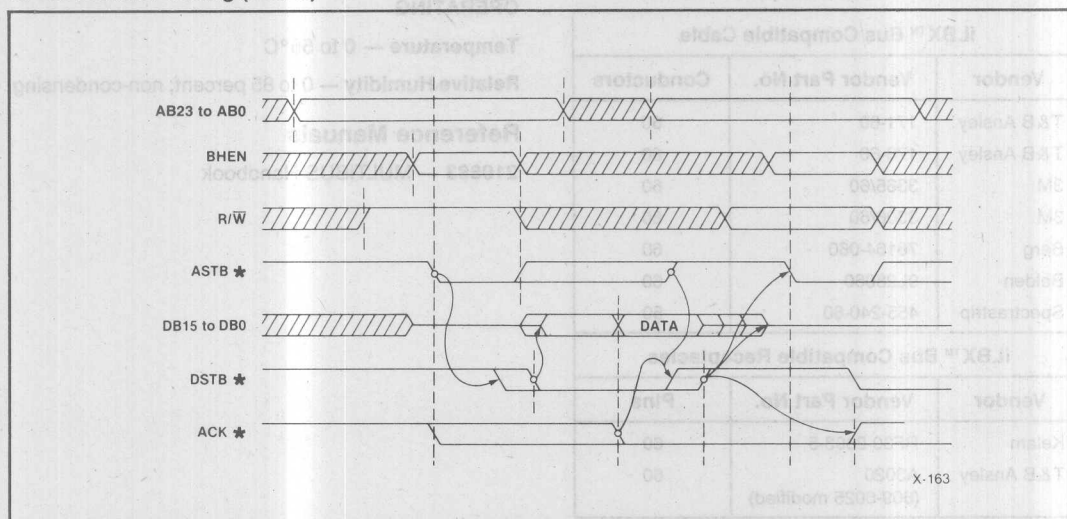


Figure 6. Read Data-From-Memory

Physical Characteristics

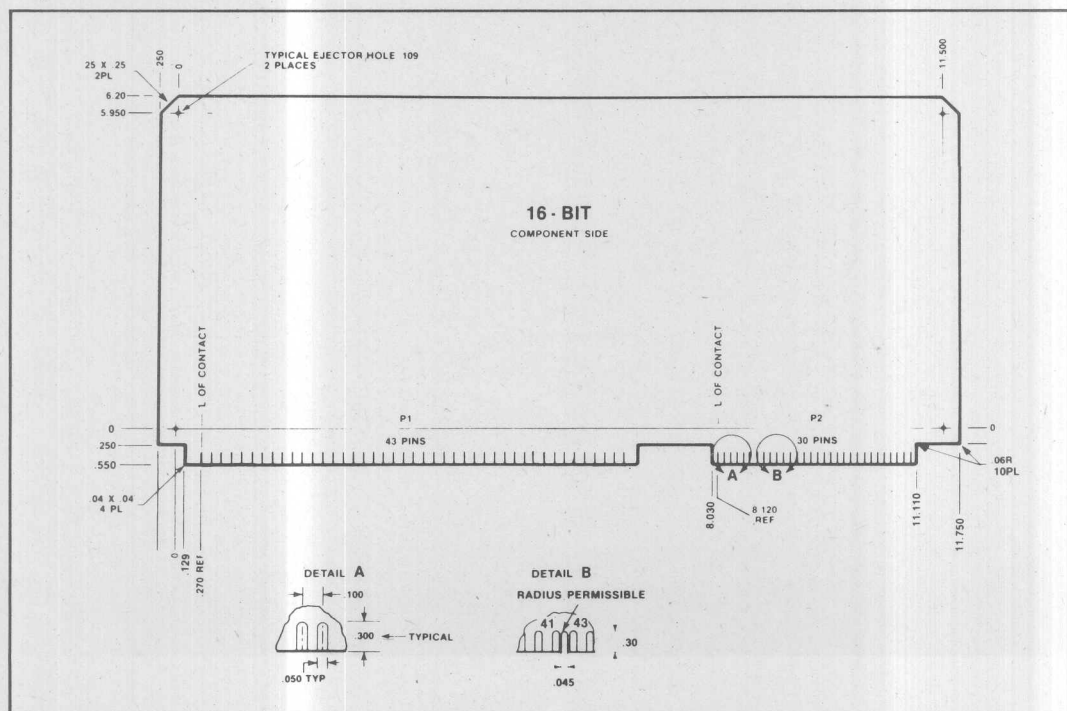


Figure 7. iLBX™ Bus Standard Printed Circuit Board Outline

Cables and Connectors

Table 3. Cable and Receptacle Vendors

iLBX™ Bus Compatible Cable		
Vendor	Vendor Part No.	Conductors
T & B Ansley	171-60	60
T & B Ansley	173-60	60
3M	3365/60	60
3M	3306/60	60
Berg	76164-060	60
Belden	9L28060	60
Spectrastrip	455-240-60	60

iLBX™ Bus Compatible Receptacles		
Vendor	Vendor Part No.	Pins
Kelam	RF30-2803-5	60
T & B Ansley	A3020 (609-6025 modified)	60

Environmental Characteristics

OPERATING

Temperature — 0 to 55°C

Relative Humidity — 0 to 85 percent; non-condensing

Reference Manuals

210883 — MULTIBUS Handbook

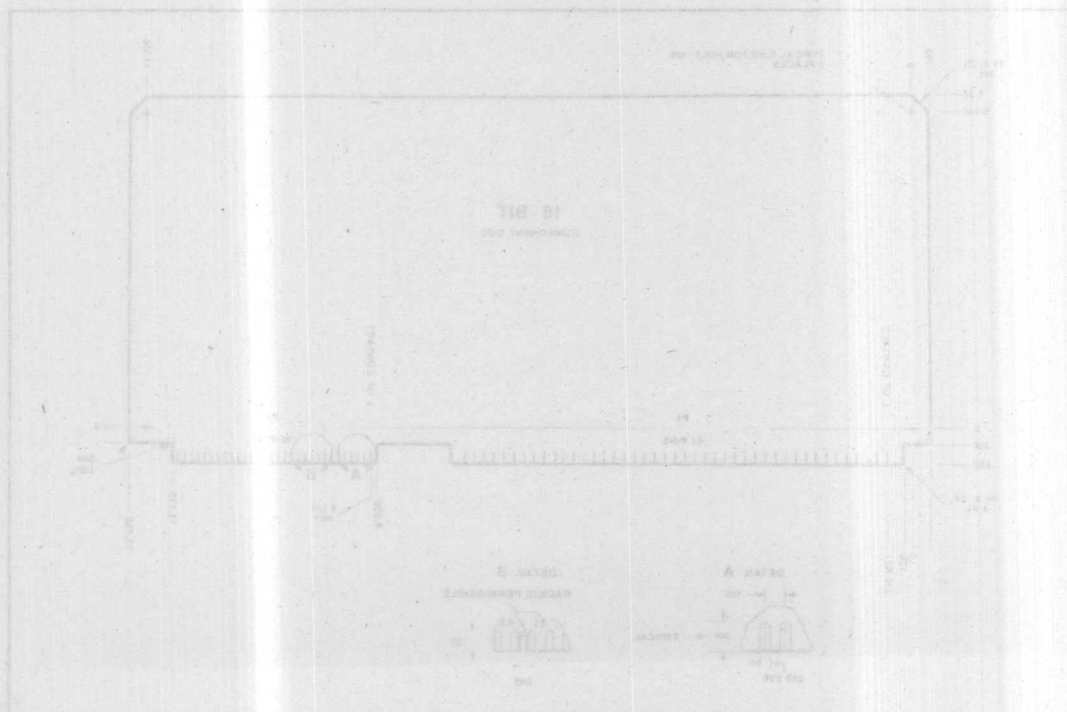
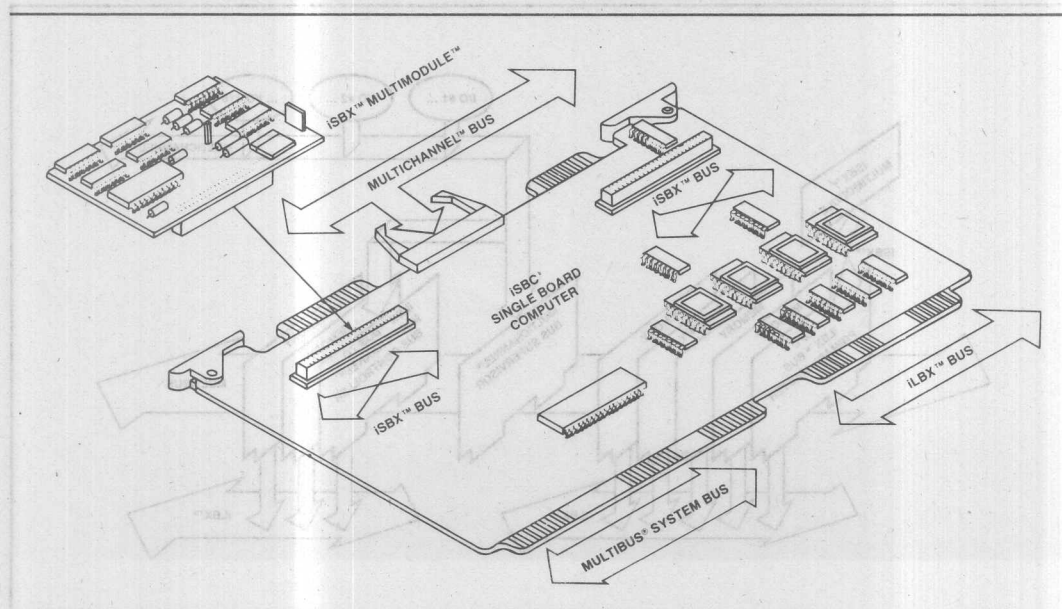


Figure 7. iLBX™ Bus Standard Printed Circuit Board Outline

ISBX™ I/O EXPANSION BUS

- IEEE P959 industry standard I/O expansion bus
- Provides on-board expansion of system resources
- Small iSBX™ MULTIMODULE™ boards plug directly into iSBC® boards
- Supports compatible 8- and 16-bit data transfer operations
- Part of Intel's Total System Architecture: MULTIBUS®, iLBX™, MULTICHANNEL™ and iSBX™
- Low-cost "vehicle" to incorporate the latest VLSI technology into iSBC®-based systems
- Provides increased functional capability and high performance
- Supported by a complete line of iSBC® base boards and iSBX™ MULTIMODULE™ boards, providing analog and digital I/O, high-speed math, serial and parallel I/O, video graphics, and peripheral controllers

The iSBX™ I/O Expansion Bus is one of a family of standard bus structures resident within Intel's total system architecture. The iSBX bus is a modular, I/O expansion bus capable of increasing a single board computer's functional capability and overall performance by providing a structure to attach small iSBX MULTIMODULE™ boards to iSBC® base boards. It provides for rapid incorporation of new VLSI into iSBC MULTIBUS® systems, reducing the threat of system obsolescence. The iSBX bus offers users new economics in design by allowing both system size and system cost to be kept at minimum. As a result, the system design achieves maximum on-board performance while allowing the MULTIBUS interface to be used for other system activities. The iSBX bus enables users to add-on capability to a system as the application demands it by providing off-the-shelf standard MULTIMODULE boards in the areas of graphics controllers, advanced mathematics functions, parallel and serial I/O, disk and tape peripheral controllers, and magnetic bubble memory. A full line of MULTIBUS boards and iSBX MULTIMODULE boards are available from Intel and other third party sources in the industry.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, IMMX, IRMX, iSBC, iSBX, ISXM, MULTIBUS, Multichannel and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supercedes previously published specifications on these devices from Intel.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Bus Elements

The ISBX™ MULTIMODULE™ system is made up of two basic elements: base boards and iSBX MULTIMODULE boards. In an iSBX system, the role of the base board is simple. It decodes I/O addresses and generates the chip selects for the iSBX MULTIMODULE boards.

The iSBX bus supports two classes of base boards, those with direct memory access (DMA) support and those without. Base boards with DMA support have DMA controllers that work in conjunction with an iSBX MULTIMODULE board (with DMA capability) to perform direct I/O to memory or memory to I/O operations. Base boards without DMA support use a subset of the iSBX bus and simply do not use the DMA feature of the iSBX MULTIMODULE board.

The iSBX MULTIMODULE boards are small, specialized, I/O mapped boards which plug into base boards. The iSBX boards connect to the iSBX bus connector and convert iSBX bus signals to a defined I/O interface.

Bus Interface/Signal Line Descriptions

The iSBX bus interface can be grouped into six functional classes: control lines, address and chip select lines, data lines, interrupt lines, option lines, and power lines. The iSBX bus provides nine control lines that de-

fine the communications protocol between base board and iSBX MULTIMODULE boards. These control lines are used to manage the general operation of the bus by specifying the type of transfer, the coordination of the transfer, and the overall state of the transfer between devices. The five address and chip select signal lines are used in conjunction with the command lines to establish the I/O port address being accessed, effectively selecting the proper iSBX MULTIMODULE. The data lines on the iSBX bus can number 8 or 16, and are used to transmit or receive information to or from the iSBX MULTIMODULE ports. Two interrupt lines are provided to make interrupt requests possible from the iSBX board to the base board. Two option lines are reserved on the bus for unique user requirements, while several power lines provide +5 and ± 12 volts to the iSBX boards.

Bus Pin Assignments

The iSBX bus uses widely available, reliable connectors that are available in 18/36 pin for 8-bit devices and 22/44 pin for 16-bit devices. The male iSBX connector is attached to the iSBX MULTIMODULE board and the female iSBX connector is attached to the base board. Figure 2 shows the dimensions and pin numbering of the 18/36 pin iSBX connector, while Figure 3 does the same for the 22/44 pin iSBX connector. A unique scheme allows the 16-bit female connector to support 8 or 16-bit male MULTIMODULE boards. Table 1 lists the signal/pin assignments for the bus.

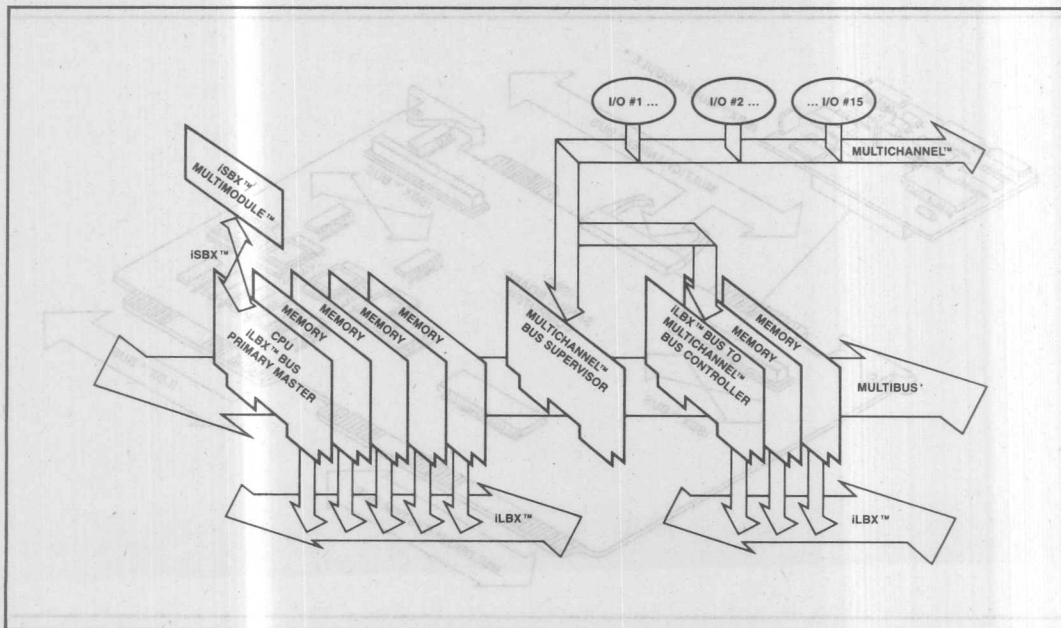


Figure 1. MULTIBUS® System Architecture

Table 1. iSBX™ Signal/Pin Assignments

Pin ¹	Mnemonic	Description	Pin ¹	Mnemonic	Description
43	MD8	MDATA Bit 8	44	MD9	MDATA Bit 9
41	MDA	MDATA Bit A	42	MDB	MDATA Bit B
39	MDC	MDATA Bit C	40	MDD	MDATA Bit D
37	MDE	MDATA Bit E	38	MDF	MDATA Bit F
35	GND	Signal Gnd	36	+5V	+5 Volts
33	MD0	MDATA Bit 0	34	MDRQT	M DMA Request
31	MD1	MDATA Bit 1	32	MDACK/	M DMA Acknowledge
29	MD2	MDATA Bit 2	30	OPT0	Option 0
27	MD3	MDATA Bit 3	28	OPT1	Option 1
25	MD4	MDATA Bit 4	26	TDMA	Terminate DMA
23	MD5	MDATA Bit 5	24		Reserved
21	MD6	MDATA Bit 6	22	MCS0/	M Chip Select 0
19	MD7	MDATA Bit 7	20	MCS1/	M Chip Select 1
17	GND	Signal Gnd	18	+5V	+5 Volts
15	IORD/	I/O Read Cmd	16	MWAIT/	M Wait
13	IOWRT/	I/O Write Cmd	14	MINTR0	M Interrupt 0
11	MA0	M Address 0	12	MINTR1	M Interrupt 1
9	MA1	M Address 1	10		Reserved
7	MA2	M Address 2	8	MPST/	iSBX Multimodule Board Present
5	RESET	Reset	6	MCLK	M Clock
3	GND	Signal Gnd	4	+5V	+5 Volts
1	+12V	+12 Volts	2	-12V	-12 Volts

Notes:

1. Pins 37-44 are used only on 8/16-bit systems
2. All undefined pins are reserved for future use.

Bus Operation Protocol

COMMAND OPERATION

The iSBX bus supports two types of transfer operations between iSBX elements: I/O Read and I/O Write. An iSBX board can respond to these I/O transfers using either full speed mode or extended mode.

For a full speed I/O Read (Figure 4) the base board generates a valid I/O address and a valid chip select for the iSBX MULTIMODULE board. After set-up, the base board activates the I/O Read line causing the iSBX board

to generate valid data from the addressed I/O port. The base board then reads the data and removes the read command, address, and chip select. The full speed I/O Write (Figure 5) operation is similar to the I/O Read except that the base board generates valid data on the lines and keeps the write command line active for the specified a hold time.

The extended Read operation (Figure 6) is used by iSBX MULTIMODULE boards that aren't configured to meet full speed specifications. It's operation is similar to full speed mode, but must use a wait signal to ensure proper

data transfer. The base board begins the operation by generating a valid I/O address and chip select. After setup, the base board activates the Read line causing the iSBX board to generate a Wait signal. This causes the CPU on the base board to go into a wait state. When the iSBX board has placed valid Read data on the data lines, the MULTIMODULE board will remove the Wait signal and release the base board CPU to read the data and deactivate the command, address, and chip select. The extended Write operation (Figure 7) is similar to the extended Read except that the Wait signal is generated after the base board places valid Write data on the data lines. The iSBX board removes the Wait signal when the write pulse width requirements are satisfied, and the base board can then remove the write command after the hold time is met.

DMA OPERATION

An iSBX MULTIMODULE system can support DMA when the base board has a DMA controller and the iSBX MULTIMODULE board can support DMA mode. Burst mode DMA is fully supported, but for clarity and simplicity, only a single DMA transfer for an 8-bit base board is discussed.

A DMA cycle (Figure 8) is initiated by the iSBX board when it activates the DMA request line going to the DMA controller on the base board. When the DMA controller gains control of the base board bus, it acknowledges back to the iSBX board and activates an I/O or Memory Read. The DMA controller then activates an I/O or Memory Write respectively. The iSBX board removes the DMA request during the cycle to allow completion of the DMA cycle. Once the write operation is complete, the DMA controller is free to deactivate the write and read command lines after a data hold time.

INTERRUPT OPERATION

The iSBX MULTIMODULE board on the iSBX bus can support interrupt operations over its interrupt lines. The iSBX board initiates an interrupt by activating one of its two interrupt lines which connect to the base board. The CPU processes the interrupt and executes the interrupt service routine. The interrupt service routine signals the iSBX MULTIMODULE board to remove the interrupt, and then returns control to the main line program when the service routine is completed.

Please refer to the Intel iSBX Bus Specification for more detailed information on its operation and implementation.

SPECIFICATIONS

Word Size

Data — 8, 16-bit

Power Supply Specifications

Table 3.

Minimum (volts)	Nominal (volts)	Maximum (volts)	Maximum (current)*
+4.75	+5.0	+5.25	3.0A
+11.4	+12	+12.6	1.0A
-12.6	-12	-11.4	1.0A
—	GND	—	3.0A

* Per iSBX Multimodule board mounted on base board.

Port Assignments

Table 2. iSBX™ MULTIMODULE™ Base Board Port Assignments

iSBX™ Connector Number	Chip Select	8-Bit Base Board Address	16-Bit Base Board Address (8-bit mode)	16-Bit Base Board Address (16-bit mode)
iSBX™ 1	MCS0/ MCS1/	F0-F7 F8-FF	0A0-0AF 0B0-0BF	0A0, 2, 4, 6, 8, A, C, E 0A1, 3, 5, 7, 9, B, D, F
iSBX™ 2	MCS0/ MCS1/	C0-C7 C8-CF	080-08F 090-09F	080, 2, 4, 6, 8, A, C, E 081, 3, 5, 7, 9, B, D, F
iSBX™ 3	MCS0/ MCS1/	B0-B7 B8-BF	060-06F 060-06F	060, 2, 4, 6, 8, A, C, E 061, 3, 5, 7, 9, B, D, F

DC Specifications

Table 4. iSBX™ MULTIMODULE™ Board I/O DC Specifications

Output¹

Bus Signal Name	Type ² Drive	I _{OL} Max —Min (mA)	@ Volts (V _{OL} Max)	I _{OH} Max —Min (μA)	@ Volts (V _{OH} Min)	C _O (Min) (pf)
MD0-MDF	TRI	1.6	0.5	—200	2.4	130
MINTR0-1	TTL	2.0	0.5	—100	2.4	40
MDRQT	TTL	1.6	0.5	— 50	2.4	40
MWAIT/	TTL	1.6	0.5	— 50	2.4	40
OPT1-2	TTL	1.6	0.5	— 50	2.4	40
MPST/	TTL	Note 3				

Input¹

Bus Signal Name	Type ² Receiver	I _{IL} Max (mA)	@ V _{IN} MAX (volts) Test Cond.	I _{IH} Max (μA)	@ V _{IN} MAX (volts) Test Cond.	C _I Max (pf)
MD0-MDF	TRI	—0.5	0.4	70	2.4	40
MA0-MA2	TTL	—0.5	0.4	70	2.4	40
MCS0/-MCS1/	TTL	—4.0	0.4	100	2.4	40
MRESET	TTL	—2.1	0.4	100	2.4	40
MDACK/	TTL	—1.0	0.4	100	2.4	40
IORD/ IOWRT/	TTL	—1.0	0.4	100	2.4	40
MCLK	TTL	— 2.0	0.4	100	2.4	40
OPT1-OPT2	TTL	—2.0	0.4	100	2.4	40

NOTES:

1. Per iSBX Multimodule I/O board.
2. TTL = standard totem pole output. TRI = Three-state.
3. iSBX Multimodule board must connect this signal to ground.

All Inputs: Max V_{IL} = 0.8V
Min V_{IH} = 2.0V

Connectors

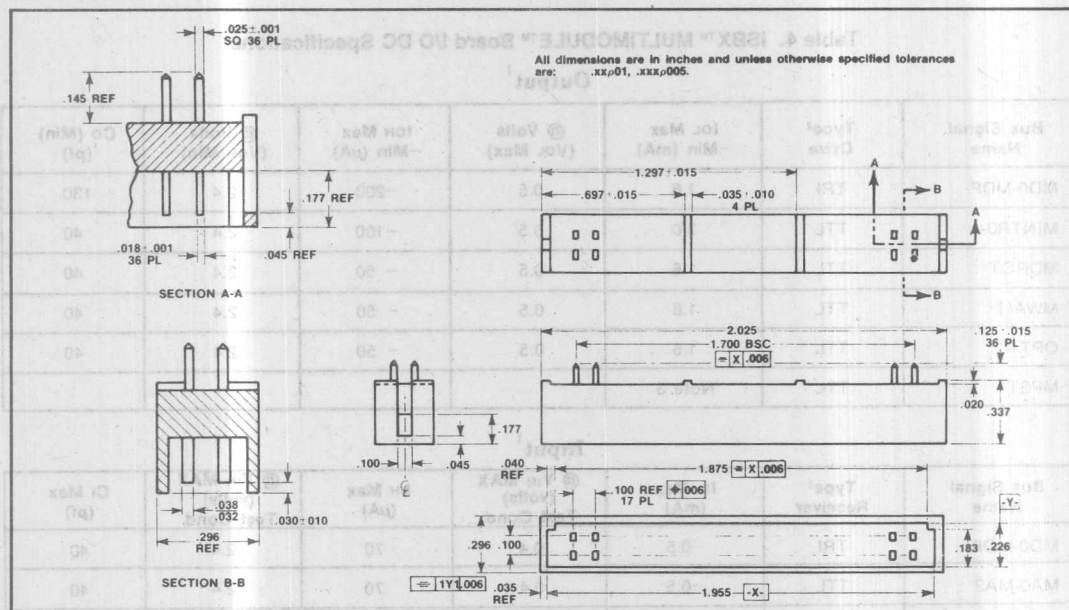


Figure 2. 18/36 Pin iSBX™ Connector

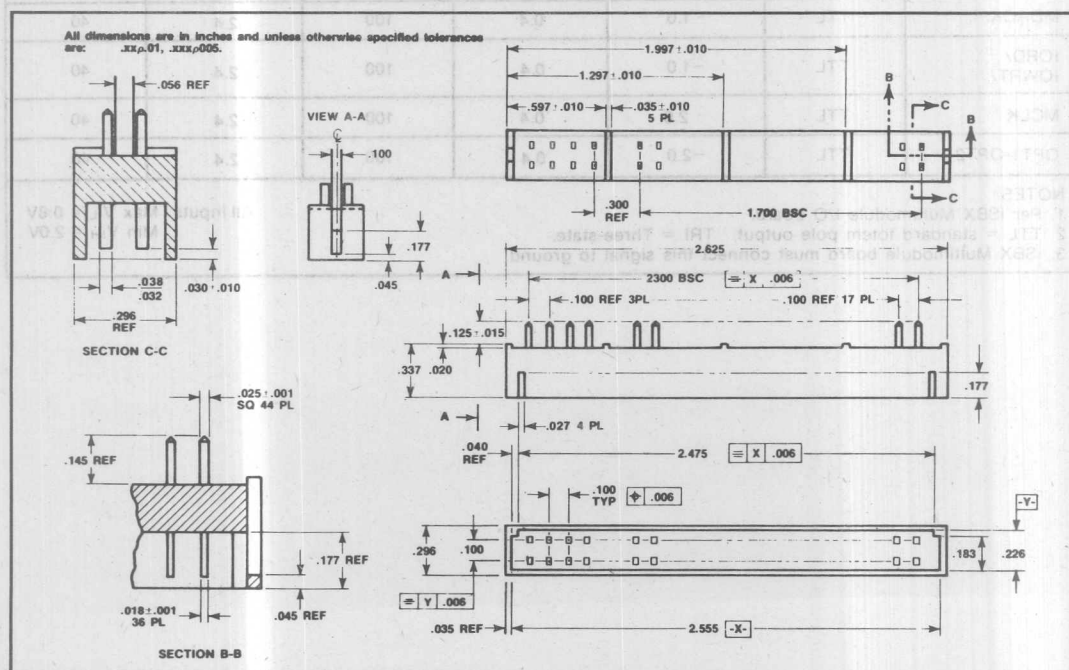


Figure 3. 22/44 Pin iSBX™ Connector

Bus Timing Diagrams

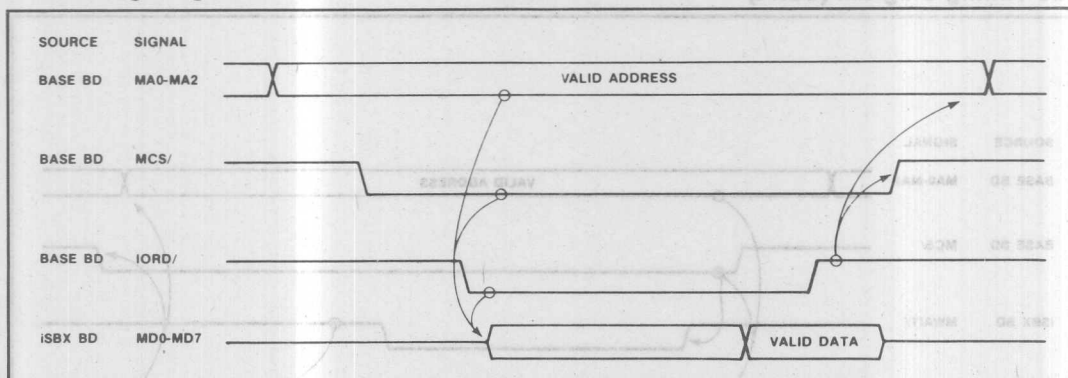


Figure 4. iSBX™ MULTIMODULE™ Read, Full Speed

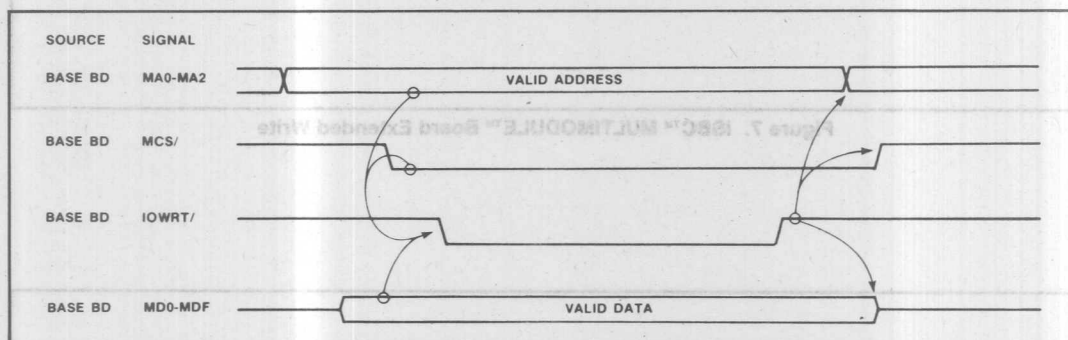


Figure 5. iSBX™ MULTIMODULE™ Board Write, Full Speed

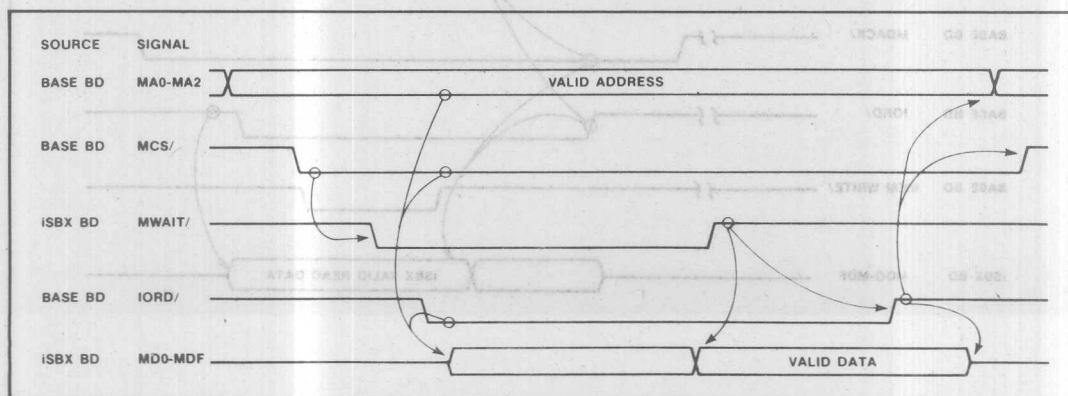


Figure 6. iSBX™ MULTIMODULE™ Board Extended Read

Bus Timing Diagram (Con't)

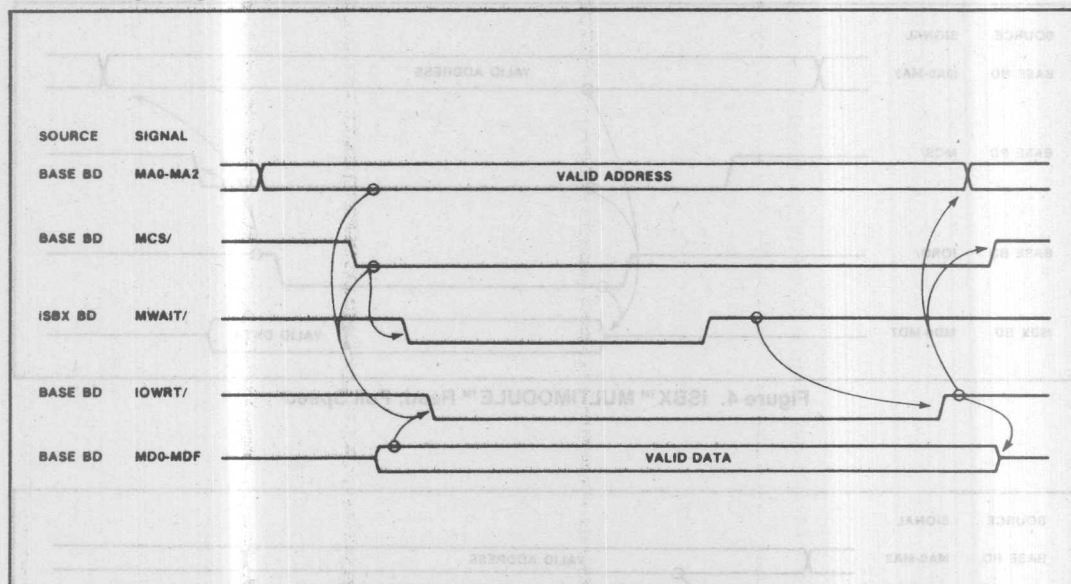


Figure 7. ISBC™ MULTIMODULE™ Board Extended Write

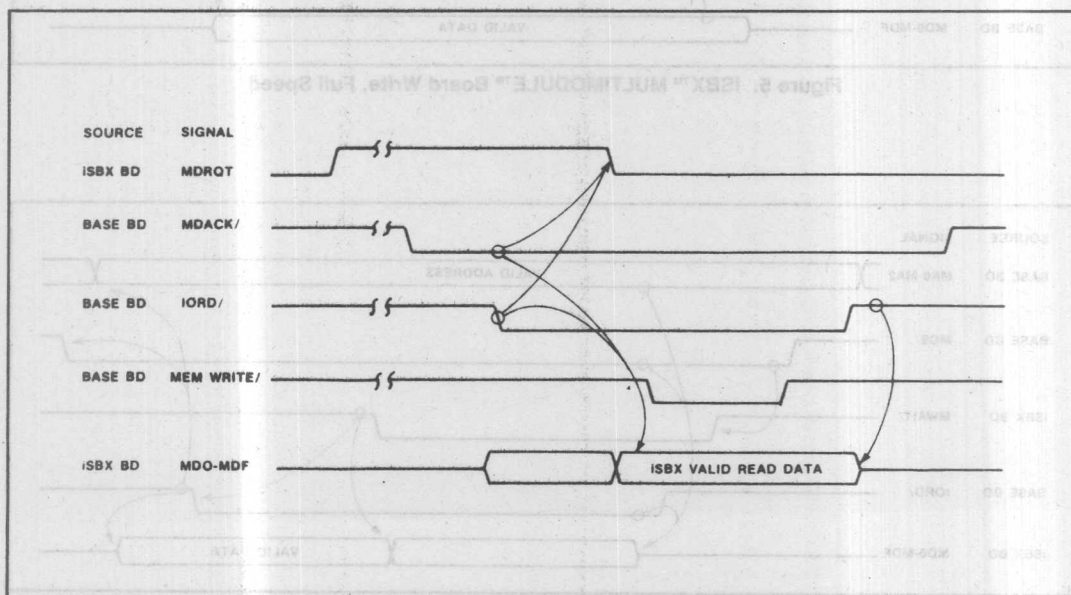


Figure 8. ISBX™ MULTIMODULE™ Board DMA Cycle (ISBX™ MULTIMODULE™ to Base Board Memory)

Board Outlines

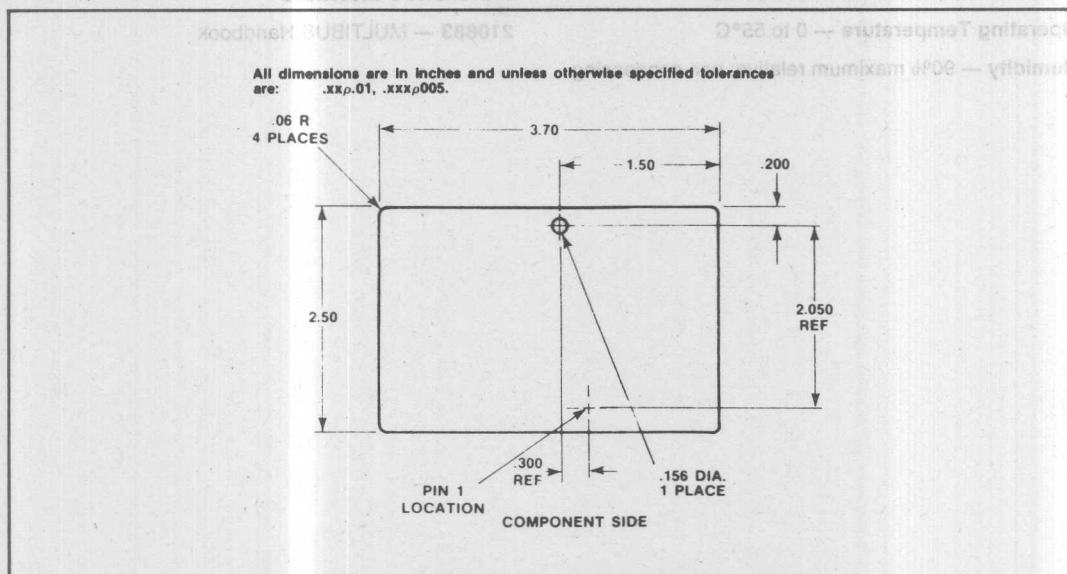


Figure 9. iSBX™ Board Outline

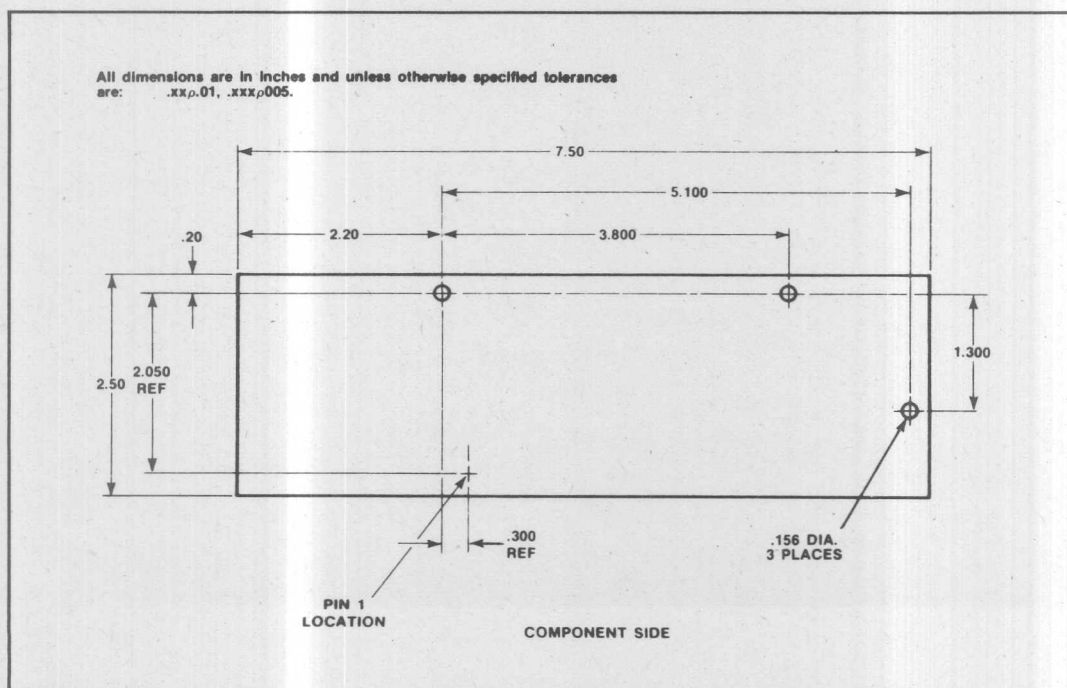


Figure 10. Double Wide iSBX™ Board Outline

Reference Manuals

210883 — MULTIBUS Handbook

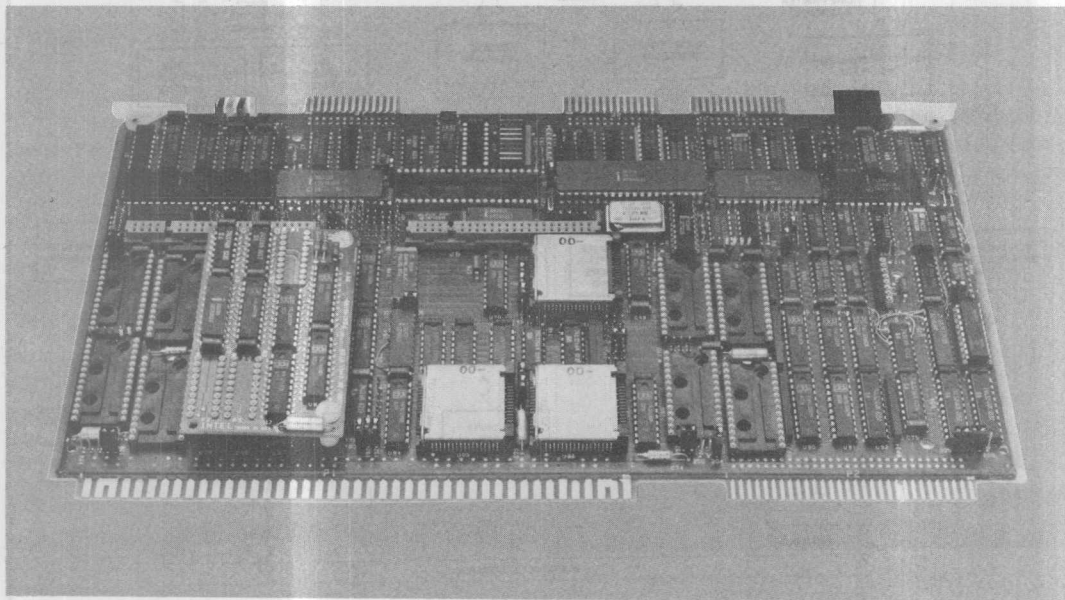
Humidity — 90% maximum relative; non-condensing



iSBC® 286/10 SINGLE BOARD COMPUTER

- iAPX 286/10 (80286) Microprocessor with 6.0 MHz CPU clock
- Optional 80287 Numeric Data Processor
- iLBX™ (Local Bus Extension) interface for high-speed memory expansion
- Two iSBX™ bus interface connectors for I/O expansion
- Eight JEDEC 28-pin sites for optional RAM/iRAM/EPROM/E²PROM components
- Optional expansion to twelve JEDEC 28-pin sites with an iSBC® 341 28-pin site expansion board
- 16 levels of vectored interrupt control
- Centronics-compatible parallel I/O printer interface
- Two programmable multiprotocol synchronous/asynchronous serial interfaces; one RS232C, the other RS232C or RS422 compatible
- MULTIBUS® interface for multimaster configurations and system expansion

The iSBC® 286/10 Single Board Computer is a member of Intel's complete line of microcomputer modules and systems which take advantage of Intel's VLSI technology to provide economical, off-the-shelf, computer-based solutions for OEM applications. The board is a complete microcomputer system on a 6.75 x 12.0 inch printed circuit card. The CPU, system clock, memory sockets, I/O ports and drivers, serial communications interface, priority interrupt logic and programmable timers, all reside on the board. The iSBC 286/10 board is the first single board computer to incorporate the iAPX 286 CPU and the iLBX™ bus extension. This combination provides the highest performance 16-bit microcomputer system solution. The iLBX architectural expansion maintains this high performance for applications requiring vast amounts of system memory.



FUNCTIONAL DESCRIPTION

Overview

The iSBC 286/10 board utilizes the powerful iAPX 286 CPU within the MULTIBUS system architecture, enhanced by the iLBX bus, to provide a high performance 16-bit solution. This board also includes on-board interrupt, memory and I/O features facilitating a complete single board computer system.

Central Processing Unit

The central processor for the iSBC 286/10 board is the 80286 CPU operating at a 6.0 MHz clock rate. The 80286 CPU is upwardly compatible with Intel's iAPX 88 and iAPX 86 CPUs. The 80286 CPU runs iAPX 88 and 86 code at substantially higher speeds due to a parallel chip architecture. In addition, the 80286 CPU provides on chip memory management and protection and virtual memory addressing of up to 1 gigabyte per task. Numeric processing power may be enhanced with the optional 80287 numerics processor. The clock rates for the 80286 and the 80287 are independent with the 80287 rate jumper selectable at either 4.0 or 8.0 MHz.

Instruction Set

The 80286 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions.

For enhanced numerics processing capability, the 80287 Numeric Data Processor extends the 80286 architecture and data set. Over 60 numeric instructions offer arithmetic, trigonometric, transcendental, logarithmic and exponential instructions. Supported data types include 16-, 32-, and 64-bit integer, 32- and 64-bit floating point, 18-digit packed BCD and 80-bit temporary. The 80287 meets the proposed IEEE P754 standard for numeric data processing and maintains compatibility with 8087-based systems.

Architectural Features

The iAPX 86, 88, 186, and 286 CPU family all contain the same basic set of registers, instructions, and addressing modes. The 80286 processor is upward compatible with the 8086, 8088 and 80186 CPUs.

The 80286 operates in two modes: iAPX 86 real address mode and protected virtual address mode. In iAPX 86

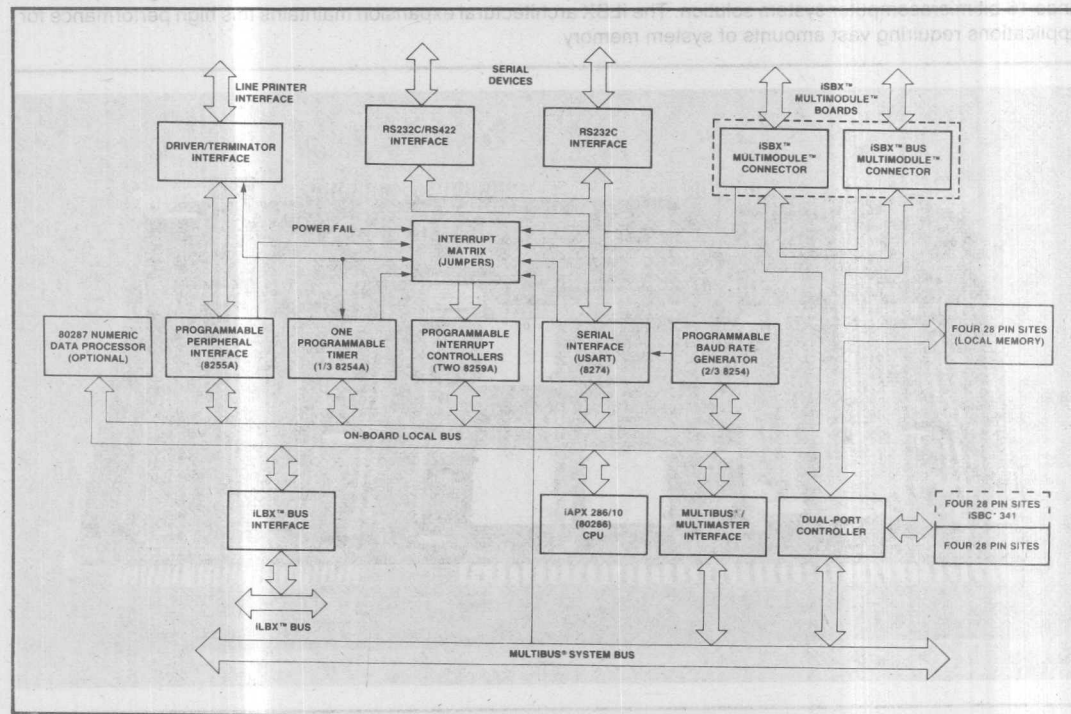


Figure 1. iSBC® 286/10 Block Diagram

real address mode, programs use real addresses with up to one megabyte of address space. Programs use virtual addresses in protected virtual address mode, also called protected mode. In protected mode, the 80286 CPU automatically maps 1 gigabyte of virtual addresses per task into a 16 megabyte real address space. This mode also provides memory protection to isolate the operating system and ensure privacy of each task's programs and data. Both modes provide the same base instruction set, registers, and addressing modes.

VECTORED INTERRUPT CONTROL

Incoming interrupts are handled by two on-board 8259A programmable interrupt controllers and by the 80286's NMI line. Interrupts originating from up to 16 sources are prioritized and then sent to the CPU as a vector address. Further interrupt capability is available through bus vectored interrupts where slave 8259 interrupt controllers resident on separate SBC boards and are cascaded into the on-board interrupt control.

INTERRUPT SOURCES

Twenty-three potential interrupt sources are routed to the interrupt jumper matrix where the user can connect the desired interrupt sources to specific interrupt levels. Table 1 includes a list of devices and functions supported by interrupts.

MEMORY CAPABILITIES

There are eight 28-pin JEDEC sites on-board which may contain a combination of byte-wide devices including RAM, iRAM, EPROM, and E²PROM. These sites are organized into two 4-site blocks, one of which may be dual-ported. The dual port block may be extended to eight sites (i.e. 12 sites total) by the addition of an iSBC 341 JEDEC site expansion module. The on-board EPROM capacity using twelve 27128 EPROMs is 192 Kbytes. The on-board RAM using ten 8K x 8 RAMs is 80 Kbytes.

SERIAL I/O

A two channel serial communications interface using Intel's 8274 Multi-Protocol Serial Controller (MPSC) is contained on the iSBC 286/10 board. Two independent software selectable baud rate generators provide the MPSC with all common communication frequencies. The protocol (i.e. asynchronous, IBM bisync, or SDLC/HDLC), data format, control character format, parity and baud rate are all under program control. Software interfacing to the MPSC can be via either a polled or interrupt driven routine. One channel may be configured for an RS232C or RS422 interface with the other channel RS232C only. The data, command and signal ground lines for each channel are brought out to two 26-pin connectors.

Table 1. Interrupt Request Sources

Device	Function	Number of Interrupts
MULTIBUS® interface	Requests from MULTIBUS® resident peripherals or other CPU boards	8*
8259A programmable interrupt controller	8 level vectored interrupt request cascaded to master 8259A	1
8274 serial controller	8 level vectored interrupt request cascaded to master 8259A	1
8255A line printer interface	Signals output buffer empty	1
8254 timers	Timer 0, 1 outputs; function determined by timer mode	2
iSBX™ connectors	Function determined by iSBX™ MULTIMODULE™ board	4 (2 per iSBX™ connector)
Bus fail safe timer	Indicates addressed MULTIBUS® resident device has not responded to command within 6 msec	1
Power fail interrupt	Indicates AC power is not within tolerance	1
External interrupt	General purpose interrupt from auxiliary connector, commonly used as front panel interrupt	1
On-board logic	Conditioned interrupt source from edge sense latch, inverter, or OR gate	3

* May be expanded to 56 with slave 8259A PICs on MULTIBUS® boards

PROGRAMMABLE TIMERS

The iSBC 286/10 board provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8254 Programmable Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller or to the 8274 MPSC to count external events or provide baud rate generation. The third interval timer in the 8254 is dedicated to providing a clock for the programmable baud rate generator in the iSBC 286/10 board's MPSC serial controller. The system software configures each timer independently to select the desired function. Seven functions are available as shown in Table 2. The contents of each counter may be read at any time during system operation.

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counter "window" has been enabled or an interrupt may be generated after N events occur in the system.

LINE PRINTER INTERFACE

An 8255A Programmable Peripheral Interface (PPI) provides a line printer interface, several on-board functions, and four non-dedicated input bits. Drivers are provided for a complete Centronics compatible line printer interface. The on-board functions implemented with the PPI are power fail sense, override, NMI mask, non-volatile RAM enable, clear timeout interrupt, LED 0 and 1, clear edge sense flop, MULTIBUS interrupt, and serial channel A loopback. The PPI's I/O lines are divided into three eight bit ports: A, B and C. Four non-dedicated input bits allow the state of four user configured jumper connections to be input. The PPI must be programmed for mode 0 with ports A and C used as outputs and port B as input. A "dummy" write to port B is used to set the iSBC 286/10 board to protected mode. The parallel port bit assignment is shown in Table 3.

Table 3. Parallel Port Bit Assignment

Port A — Output	
Bit	Function
0	Line Printer Data Bit 0
1	Line Printer Data Bit 1
2	Line Printer Data Bit 2
3	Line Printer Data Bit 3
4	Line Printer Data Bit 4
5	Line Printer Data Bit 5
6	Line Printer Data Bit 6
7	Line Printer Data Bit 7
Port B — Input	
Bit	Function
0	General Purpose Input 0
1	General Purpose Input 1
2	General Purpose Input 2
3	General Purpose Input 3
4	Line Printer ACK/ (Active Low)
5	Power Fail Sense/ (Active Low)
6	Line Printer Error (Active Hi)
7	Line Printer Busy (Active Hi)
Port C — Output	
Bit	Function
0	Line Printer Data Strobe (Active Hi)
1	Override/ (Active Low)
2	NMI Mask (0 = NMI Enabled)
3	Non-Volatile RAM Enable; Clear Timeout Interrupt/
4	LED 0 (1 = On); Clear Edge Sense Flop/
5	MULTIBUS® Interrupt (1 = Active)
6	Serial CHA Loopback (0 = Online, 1 = Loopback)
7	LED 1 (1 = On); Clear Line Printer Ack Flop/

MULTIBUS® System Architecture

OVERVIEW

The MULTIBUS system architecture includes three bus structures: the system bus, the local bus extension and the MULTIMODULE™ expansion bus as shown in Figure 2. Each bus structure is optimized to satisfy particular system requirements. The system bus provides a basis for general system design including memory and I/O expansion as well as multiprocessing support. The local bus extension allows large amounts of high performance memory to be accessed from a CPU board over a private bus. The MULTIMODULE extension bus is a means of adding inexpensive I/O functions to a base CPU board. Each of these three bus structures are implemented on the iSBC 286/10 board providing a total system architecture solution.

SYSTEM BUS – IEEE 796

The MULTIBUS system bus is Intel's industry standard, IEEE 796, microcomputer bus structure. Both 8- and 16-bit single board computers are supported on the IEEE 796 structure with 24 address and 16 data lines. In its simplest application, the system bus allows expansion of functions already contained on a single board computer (e.g., memory and digital I/O). However, the IEEE 796 bus also allows very powerful distributed pro-

cessing configurations with multiple processors and intelligent slave, I/O and peripheral boards capable of solving the most demanding microcomputer applications. The MULTIBUS system bus is supported with a broad array of board level products, LSI interface components, detailed published specifications and application notes.

SYSTEM BUS – EXPANSION CAPABILITIES

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. Memory may be expanded by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be added with digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers, or hard disk controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

SYSTEM BUS – MULTIMASTER CAPABILITIES

For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication of the system bus), the iSBC 286/10 board provides full system bus arbitration control logic. This control logic allows up to three iSBC

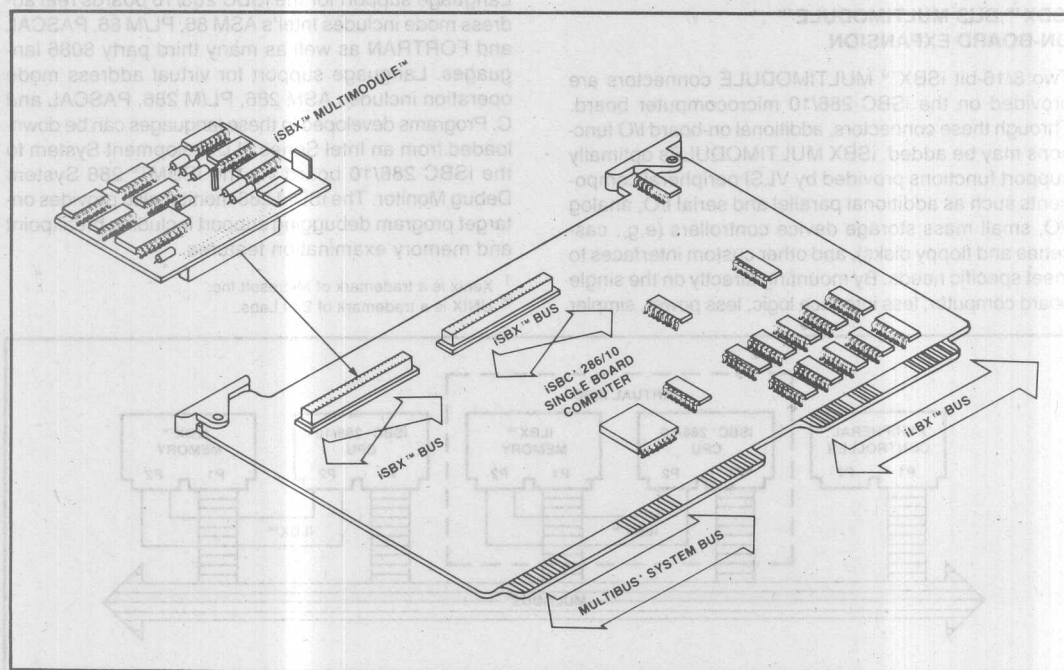


Figure 2. MULTIBUS® System Architecture

286/10 boards or other bus masters, including iSBC 80 family MULTIBUS compatible 8-bit single board computers to share the system bus using a serial (daisy chain) priority scheme and allows up to 16 masters to share the MULTIBUS system bus with an external parallel priority decoder. In addition to multiprocessing configuration made possible with multimaster capability, it also provides a very efficient mechanism for all forms of DMA (Direct Memory Access) transfers.

iLBX™ BUS – LOCAL BUS EXTENSION

The iSBC 286/10 board also provides the local bus extension (iLBX) of the MULTIBUS architecture. This standard extension allows on-board memory performance with physically off-board memory. The combination of a CPU board and iLBX memory boards is architecturally equivalent to a single board computer and thus can be called a "virtual SBC". The iLBX is implemented over the P2 connector and requires cabling across the virtual SBCs of a system (see Figure 3). Other Intel products which support the iLBX bus include:

- iSBC 028CX 128KB iLBX RAM board
- iSBC 056CX 256KB iLBX RAM board
- iSBC 012CX 512KB iLBX RAM board
- iSBC 428 JEDEC 28-PIN SITE board
- iSBC 580 MULTICHANNEL™ interface board

iSBX™ BUS MULTIMODULE™ ON-BOARD EXPANSION

Two 8/16-bit iSBX™ MULTIMODULE connectors are provided on the iSBC 286/10 microcomputer board. Through these connectors, additional on-board I/O functions may be added. iSBX MULTIMODULEs optimally support functions provided by VLSI peripheral components such as additional parallel and serial I/O, analog I/O, small mass storage device controllers (e.g., cassettes and floppy disks), and other custom interfaces to meet specific needs. By mounting directly on the single board computer, less interface logic, less power, simpler

packaging, higher performance, and lower cost result when compared to other alternatives such as MULTIBUS form factor compatible boards. The iSBX connectors on the iSBC 286/10 board provide all signals necessary to interface to the local on-board bus, including 16 data lines for maximum data transfer rates. iSBX MULTIMODULE boards designed with 8-bit data paths and using the 8-bit iSBX connector are also supported on the iSBC 286/10 microcomputer board. A broad range of iSBX MULTIMODULE options are available from Intel. Custom iSBX modules may also be designed. An iSBX bus interface specification and iSBX connectors are available from Intel.

Software Support

Real-time support for the iSBC 286/10 board is provided by the iRMX™ 286R operating system. iRMX 286R is an adaptation of iRMX 86 nucleus to operate on the iSBC 286/10 board in real address mode, enhances the ICU for configuration support of the board, adds a driver for the on-board 8274 and supports the 80287. The iRMX 286R Operating System is completely compatible with iRMX 86.

Interactive multi-user support will be provided by the XENIX¹ operating system. XENIX is a compatible derivative of UNIX², System III.

Language support for the iSBC 286/10 boards real address mode includes Intel's ASM 86, PL/M 86, PASCAL and FORTRAN as well as many third party 8086 languages. Language support for virtual address mode operation includes ASM 286, PL/M 286, PASCAL and C. Programs developed in these languages can be downloaded from an Intel Series III Development System to the iSBC 286/10 board via the iSDM™ 286 System Debug Monitor. The iSDM 286 monitor also provides on-target program debugging support including breakpoint and memory examination features.

¹ Xenix is a trademark of Microsoft Inc.

² UNIX is a trademark of Bell Labs.

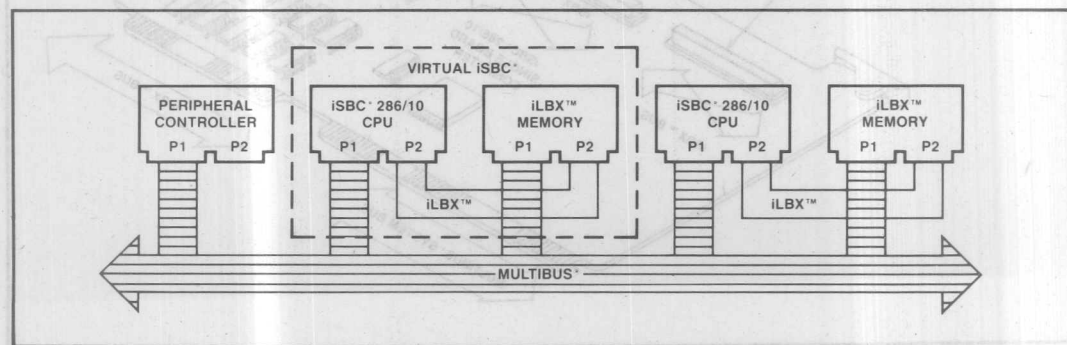


Figure 3. MULTIBUS®/iLBX™ Configuration

SPECIFICATIONS

Word Size

Instruction — 8, 16, 24, 32 or 40 bits

Data — 8 or 16 bits

System Clock

CPU — 6.0 MHz

Numeric Processor — 4.0 or 8.0 MHz (Jumper Selectable)

Cycle Time

Basic Instruction — 6.0 MHz - 500 ns; 333 ns (assumes instruction in queue)

NOTE: Basic instruction cycle is defined as the fastest instruction time (i.e. two clock cycles).

Memory Capacity (Maximum)

EPROM — 2716, 8 Kbytes; 2732, 16 Kbytes; 2764, 64 Kbytes; 27128, 128 Kbytes; 27256, 256 Kbytes

E²PROM — 2817A, 16 Kbytes

iRAM — 2186, 16 Kbytes

Static RAM — 8K x 8 devices, 48 Kbytes

NOTES: Two local sites must contain boot-up EPROM or E²PROM. 2716s and 2732s may reside in dual-port sites only. iRAMs may reside in local sites only.

WITH iSBC® 341 MULTIMODULE™

EPROM — 2716, 16 Kbytes; 2732, 32 Kbytes; 2764, 96 Kbytes; 27128, 192 Kbytes; 27256, 256 Kbytes

E²PROM — 2817A, 24 Kbytes

iRAM — 2186, 16 Kbytes

Static RAM — 8K x 8 devices, 80 Kbytes

NOTES: Dual-port sites can address 128 Kbytes of memory maximum. Two local sites must contain boot-up EPROM or E²PROM. 2716s and 2732s may reside in dual-port sites only. iRAMs may reside in local sites only.

I/O Capability

Parallel — Line printer interface, on-board functions, and four non-dedicated input bits

Serial — Two programmable channels using one 8274

Timers — Three programmable timers using one 8254

Expansion — Two 8/16-bit iSBX MULTIMODULE connectors

Interrupt Capacity

Potential Interrupt Sources — 23, jumper selectable

Interrupt Levels — 16 vectored requests using two 8259As and the 80286's NMI line.

Serial Communications Characteristics

Synchronous — 5-8 bit characters; internal or HDLC/SDLC character synchronization; automatic sync insertion; even or odd parity.

Asynchronous — 5-8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection; even or odd parity.

BAUD RATES

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)				
	Synchronous	Asynchronous			
Reference: 1.23 MHz	÷ 1	÷ 1	÷ 6	÷ 32	÷ 64
615.	615,000	615,000	38,400	19,200	9,600
307.	307,000	307,000	19,200	9,600	4,800
154.	154,000	154,000	9,600	4,800	2,400
76.8	76,800	76,800	4,800	2,400	1,200
38.4	38,400	38,400	2,400	1,200	600
19.2	19,200	19,200	1,200	600	300
9.6	9,600	9,600	600	300	150
4.8	4,800	4,800	300	150	75
2.4	2,400	2,400	150	75	—
1.2	1,200	1,200	75	—	—
0.6	600	600	—	—	—

TIMERS

Input Frequencies — 1.23 MHz \pm 0.1% or 3.00 MHz \pm 0.1% (Jumper Selectable)

OUTPUT FREQUENCIES/TIMING INTERVALS

Function	Single Timer/Counter		Dual Timer/Counter (two timers cascaded)	
	Min	Max	Min	Max
Real-time interrupt	667 ns	53.3 ms	1.33 μ s	58.2 min
Programmable one-shot	667 ns	53.3 ms	1.33 μ s	58.2 min
Rate generator	18.8 Hz	1.50 MHz	0.000286 Hz	750 kHz
Square-wave rate generator	18.8 Hz	1.50 MHz	0.000286 Hz	750 kHz
Software triggered strobe	667 ns	53.3 ms	1.33 μ s	58.2 min
Hardware triggered strobe	667 ns	53.3 ms	1.33 μ s	58.2 min
Event counter	—	8.0 MHz	—	—

INTERFACES

MULTIBUS® — All signals TTL compatible

iSBX™ Bus — All signals TTL compatible

iLBX™ Bus — All signals TTL compatible

Serial I/O — Channel A: RS232C/RS422 compatible, configurable as a data set or data terminal; Channel B: RS232C compatible, configured as data set

Timer — All signals TTL compatible

Interrupt Requests — All TTL compatible

CONNECTORS

Interface	Double-Sided Pins (qty.)	Centers (in.)	Mating Connectors
MULTIBUS® System	86	0.156	Viking 3KH43/9AMK12 Wire Wrap
iSBX™ Bus — 8-Bit Data	36	0.1	iSBX™ 960-5
16-Bit Data	44	0.1	iSBX™ 961-5
iLBX™ Bus	60	0.1	Kelam RF30-2803-5 or T&B Ansley A3020 (609-6026 modified)
Parallel I/O	26	0.1	3M 3462-0001 Flat or AMP 88106-1 Flat
Serial I/O	26	0.1	3M 3462-0001 Flat or AMP 88106-1 Flat

MULTIBUS® DRIVERS

Function	Characteristic	Sink Current (mA)
Data	Tri-State	16
Address	Tri-State	16
Commands	Tri-State	32
Bus Control	Open Collector	20

iLBX™ DRIVERS

Function	Characteristic	Sink Current (mA)
Data	Tri-State	9
Address	Tri-State	20
Commands	Tri-State	8
Bus Control	TTL	8

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.70 in. (1.78 cm)

NOTE: Depth includes a small piggyback on lower left of board.

Weight — 19 oz. (539 gm)

Electrical Characteristics

DC Power Requirements — +5V, 7.0A; +12V, 50 mA;
–12V, 50 mA

NOTE: Does not include power for optional EPROM, E²PROM, or RAM.

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Relative Humidity — to 90% (without condensation)

Reference Manual

145439-001 — iSBC 286/10 Single Board Computer
Design Guide (NOT SUPPLIED)

ORDERING INFORMATION

Part Number Description

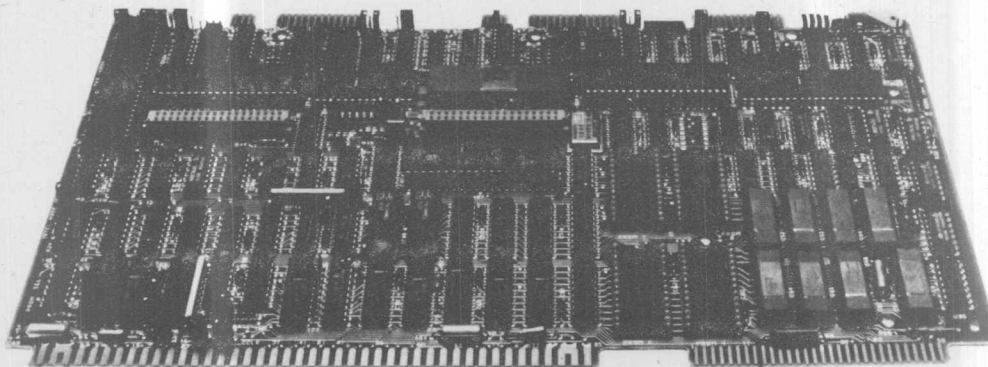
SBC 286/10 Single Board Computer



iSBC® 88/45 ADVANCED DATA COMMUNICATIONS PROCESSOR BOARD

- Three HDLC/SDLC half/full-duplex communication channels — optional ASYNC/SYNC on two channels
- Supports RS232C (including modem support), CCITT V.24, or RS422A/449 interfaces
- On-board DMA supports 800K baud operation
- Self-clocking NRZI SDLC loop data link interface
 - point-to-point
 - multidrop
- Software programmable baud rate generation
- iAPX 88/10 (8088-2) Microprocessor operates at 8 MHz
- iSBC® 337 Numeric Data Processor option supported
- 16K bytes static RAM (12K bytes dual-ported)
- Four 28-pin JEDEC sites for EPROM/RAM expansion; four additional 28-pin JEDEC sites added with iSBC® 341 board
- Two iSBX™ bus connectors
- MULTIBUS® interface supports Multimaster configuration

The iSBC 88/45 Advanced Data Communications Processor (ADCP) Board adds 8 MHz, iAPX 88/10 (8088-2) 8-bit microprocessor-based communications flexibility to the Intel line of OEM microcomputer systems. The iSBC 88/45 ADCP board offers asynchronous, synchronous, SDLC, and HDLC serial interfaces for gateway networking or general purpose solutions. The iSBC 88/45 ADCP board provides the CPU, system clock, EPROM/RAM, serial I/O ports, priority interrupt logic, and programmable timers to facilitate higher-level application solutions.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, IRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Three Communication Channels

Three programmable HDLC/SDLC serial interfaces are provided on the iSBC 88/45 ADCP board. The SDLC interface is familiar to IBM system and terminal equipment users. The HDLC interface is known by users of CCITT's X.25 packet switching interface.

One channel utilizes an Intel 8273 controller to manage the serial data transfers. Accepting the 8-bit data bytes from the local bus, the 8273 controller translates the data into the HDLC/SDLC format. The channel operates in half/full-duplex mode.

In addition to the synchronous mode, the 8273 controller operates asynchronously with NRZI encoded data which is found in systems such as the IBM 3650 Retail Store System. An SDLC loop configuration using iSBX 352 and iSBC 88/45 products is shown in Figure 1.

The two additional channels utilize the Intel 8274 Multi-Protocol Serial Controller (MPSC). The MPSC provides two independent half/full-duplex serial channels which provide asynchronous, synchronous, HDLC or SDLC protocol operations. The sync and async protocol operations are commonly used to communicate with inexpensive terminals and systems.

The three serial channels of the iSBC 88/45 ADCP board offer communications capability to manage a gateway application. The gateway application, as shown in Figure 1, manages diverse protocol requirements for data movement between channels. Typical protocol management software layers im-

plemented by the user include SNA terminal interfaces to IBM systems.

On-Board DMA

For high-speed communications, one MPSC channel has a DMA capacity to support an 800K baud rate. The second channel attached to the MPSC is capable of simultaneous 800K baud operation when configured with DMA capability, but is connected to an RS232C interface which is defined as 20K baud maximum. Figure 2 shows an RS422A/449 multidrop application which supports high-speed operation.

Interfaces Supported

The iSBC 88/45 ADCP board provides an excellent foundation to support these electrical and diverse software drivers protocol interfaces. The control lines, serial data lines, and signal ground lines are brought out to the three double-edge connectors. Figure 3 shows the cable to connector construction. Two connectors are pre-configured for RS422A/449. All three channels are configurable for RS232C/CCITT V.24 interfaces as shown in Table 1.

Table 1. iSBC® 88/45 Supported Configurations

Connection	Synchronous		Asynchronous	
	Modem	Direct	Modem*	Direct
point-to-point	X**	X	X	X
multidrop	X	X	X	X
loop	N.A.	N.A.	C (only)	C (only)

* Modem should not respond to break.

** Channels A, B, and C denoted by X.

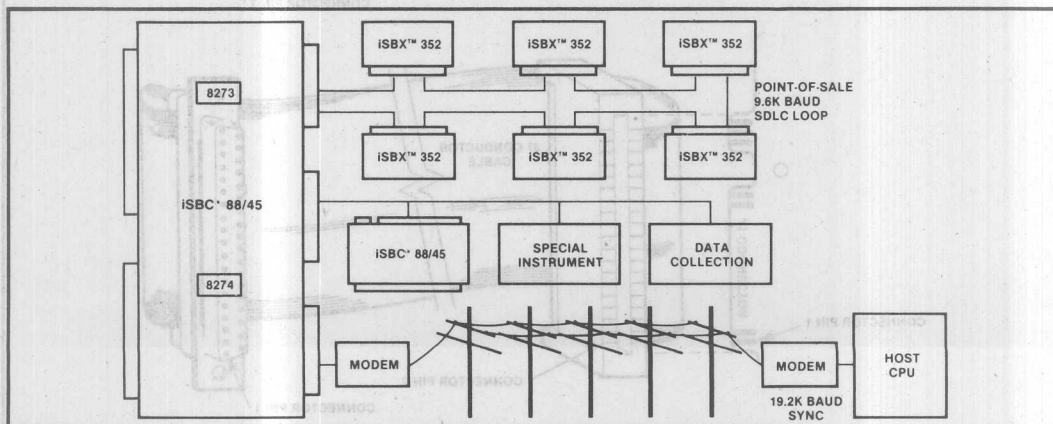


Figure 1. iSBC® 88/45 Gateway Processor Example

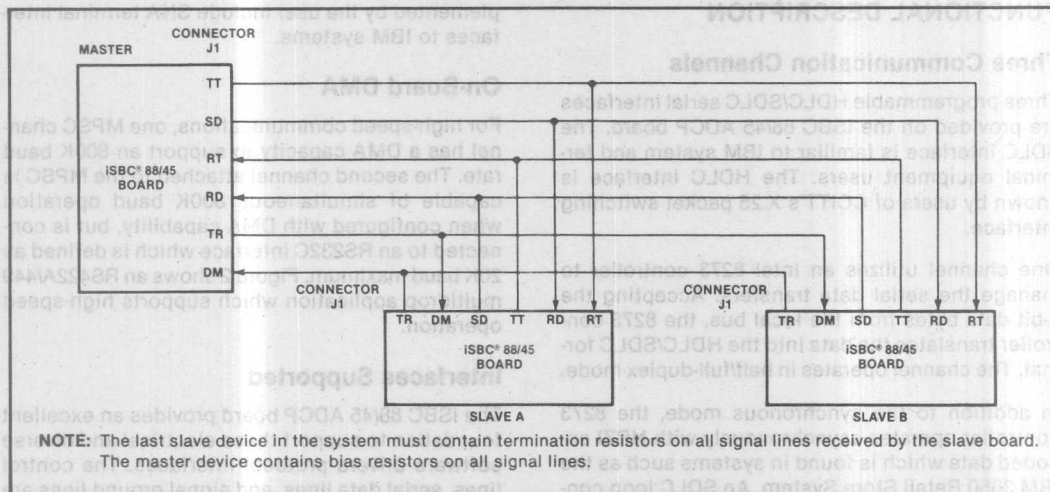


Figure 2. Synchronous Multidrop Network Configuration Example - RS422A

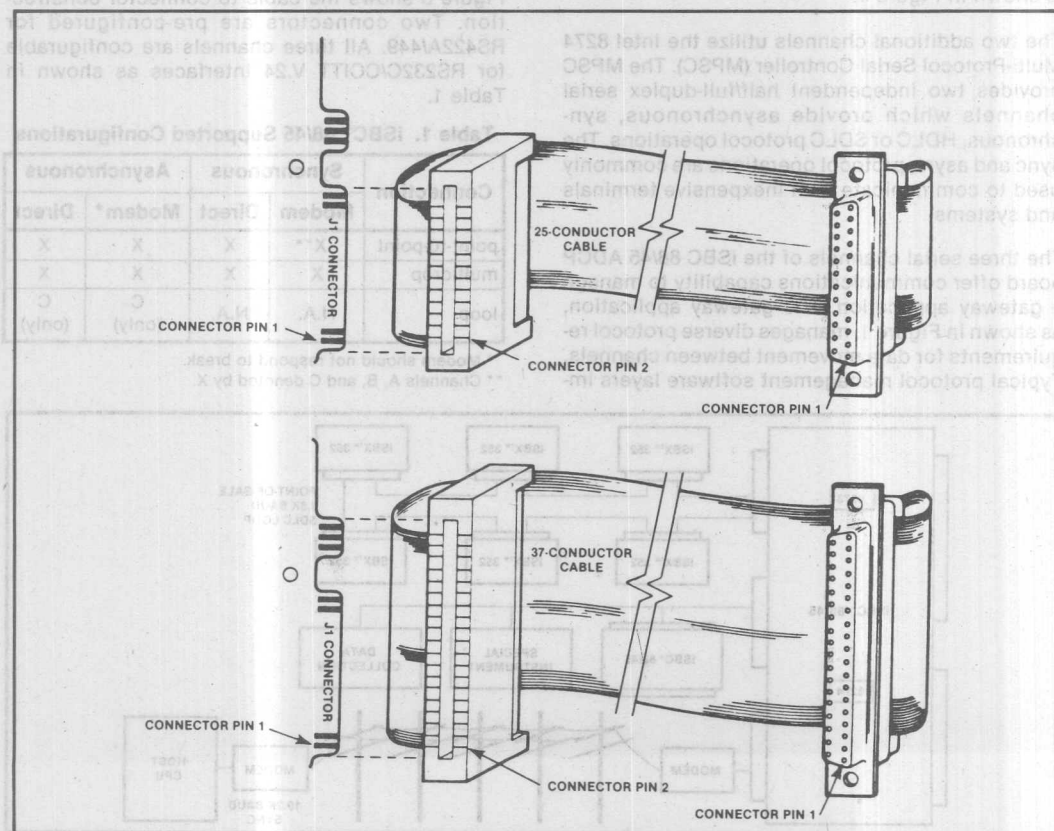


Figure 3. Cable Construction and Installation for RS232C and RS422A/449 Interface

Self Clocking Point-To-Point Interface

The iSBC 88/45 ADCP board is used in an asynchronous mode interface when configured as shown in Figure 4. The point-to-point RS232C example uses the self-clocking mode interface for NRZI encoding/decoding of data. The digital phase-lock loop allows operation of the interface in either half/duplex or full/duplex implementation with or without modems.

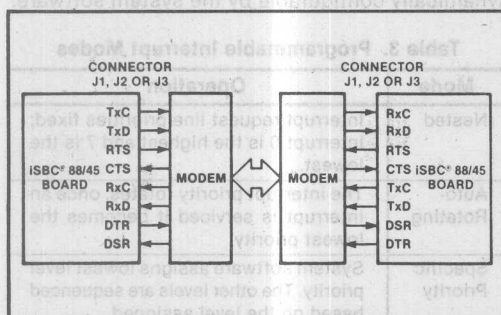


Figure 4. Self-Clocking or Asynchronous Point-to-Point Modem Interface Configuration Example - RS232C

Synchronous Point-To-Point Interface

Figure 5 shows a synchronous point-to-point mode of operation for the iSBC 88/45 ADCP board. This RS232C example uses a modem to generate the receive clock for coordination of the data transfer. The iSBC 88/45 ADCP board generates the transmit synchronizing clock for synchronous transmission.

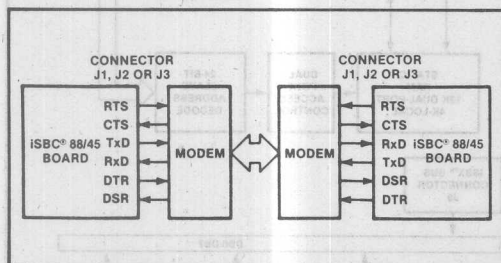


Figure 5. Synchronous Point-to-Point Modem Interface Configuration Example - RS232C

Central Processing Unit

The central processor for the iSBC 88/45 Advanced Data Communications Processor board is Intel's 8088 microprocessor operating at 8 MHz. The microprocessor interface to other functions is

illustrated in Figure 6. The microprocessor architecture is designed to effectively execute the application and networking software written in higher-level languages.

This architectural support includes four 16-bit byte addressable data registers, two 16-bit memory base pointer registers and two 16-bit index registers. These registers are addressable through 24 different operand addressing modes for comprehensive memory addressing and for high-level language data structure manipulation.

The stack-oriented architecture readily supports Intel's iRMX executives and iMMX multiprocessing software. Both software packages are designed for modular application programming. Facilitating the fast inter-module communications, the 4-byte instruction queue supports program constructs needed for real-time systems.

Since programs are segmented between pure procedure and data, four segment registers (code, stack, data, extra) are available for addressing 1 megabyte of memory space. These registers contain the offset values used to address a 64K byte segment. The registers are controlled explicitly through program control or implicitly by high-level language functions and instructions.

The real-time system software can also utilize the programmable timers as shown in Table 2 and various interrupt control modes available on the ADCP board to have responsive and effective application solutions.

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on Terminal Count	An interrupt is generated on terminal count being reached. This function is useful for generation of real-time clocks.
Rate Generator	Divide by N counter. Based on the input clock period, the output pulse remains low until the count is expired.
Square Wave Generator	Output remains high for one-half the count, goes low for the remainder of the count.
Software Triggered Strobe	Output remains high until count expires, then goes low for one clock period.

Numeric Data Processor Extension

The 8088 instruction set includes 8-bit and 16-bit signed and unsigned arithmetic operators for bi-

nary, BCD, and unpacked ASCII data. For enhanced numerics processing capability, the iSBC 337 MULTIMODULE Numeric Data Processor extends the 8088 architecture and data set¹.

The extended numerics capability includes over 60 numeric instructions offering arithmetic, trigonometric, transcendental, logarithmic, and exponential instructions. Many math-oriented applications utilize the 16-, 32-, and 64-bit integer, 32- and 64-bit floating point, 18-digit packed BCD, and 80-bit temporary data types.

16K Bytes Static Ram

The iSBC 88/45 ADCP board contains 16K bytes of high-speed static RAM, with 12K bytes dual-ported which is addressable from other MULTIBUS devices. When coupled with the high-speed DMA capability of the iSBC 88/45 ADCP board, the dual-ported memory provides effective data communication buffers. The dual-ported memory is useful for interprocessor message transfers.

¹ The iSBC 337 board requires the iSBC 88/45 ADCP board be jumpered to provide 4 MHz operation.

Interrupt Capability

The iSBC 88/45 ADCP board provides nine vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line. The additional eight interrupt levels are vectored via the Intel 8259A Programmable Interrupt Controller (PIC). As shown in Table 3, four priority processing modes are available to match interrupt servicing requirements. These modes and priority assignments are dynamically configurable by the system software.

Table 3. Programmable Interrupt Modes

Mode	Operation
Nested	Interrupt request line priorities fixed; interrupt 0 is the highest and 7 is the lowest.
Auto-Rotating	The interrupt priority rotates; once an interrupt is serviced it becomes the lowest priority.
Specific Priority	System software assigns lowest level priority. The other levels are sequenced based on the level assigned.
Polled	System software examines priority interrupt via interrupt status register.

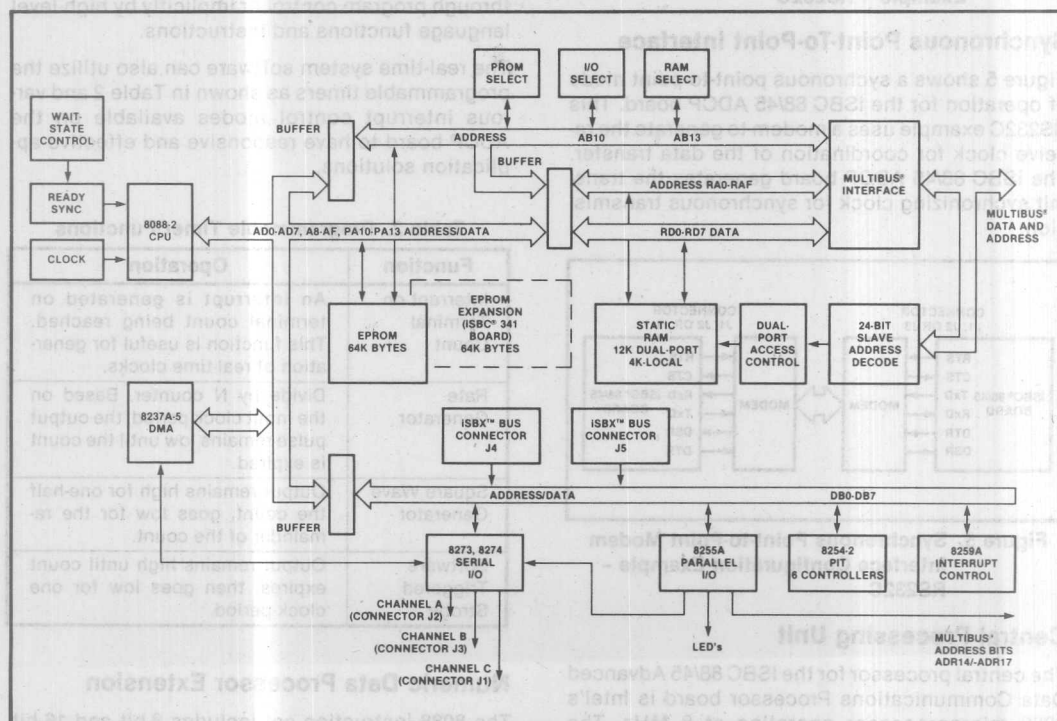


Figure 6. Block Diagram of the iSBC® 88/45 ADCP Board

Interrupt Request Generation

Listed in Table 4 are the devices and functions supported by interrupts on the iSBC 88/45 ADCP board. All interrupt signals are brought to the interrupt jumper matrix. Any of the 23 interrupt sources are strapped to the appropriate 8259A PIC request level. The PIC resolves requests according to the software selected mode and, if the interrupt is unmasked, issues an interrupt to the CPU.

EPROM/RAM Expansion

In addition to the on-board RAM, the iSBC 88/45 ADCP board provides four 28-pin JEDEC sockets for EPROM expansion. By using 2764 EPROMs, the board has 32K bytes of program storage. Three of the JEDEC standard sockets also support byte-wide static RAMs; using 8K x 8 static RAMs provides an additional 24K bytes of RAM.

Inserting the optional iSBC 341 MULTIMODULE EPROM expansion board onto the iSBC 88/45 ADCP board provides four additional 28-pin JEDEC sites. This expansion doubles the available program storage or extends the RAM capability by 32K bytes.

ISBX™ MULTIMODULE™ Expansion

Two 8-bit iSBX MULTIMODULE connectors are provided on the iSBC 88/45 microcomputer. Through these connectors, additional iSBX functions extend the I/O capability of the microcom-

puter. The iSBX connectors provide the necessary signals to interface to the local bus.

In addition to specialized or custom designed iSBX boards, the customer has a broad range of Intel iSBC MULTIMODULEs available, including parallel I/O, analog I/O, IEEE 488 GPIB, floppy disk, magnetic bubbles, video, and serial I/O boards.

The serial I/O MULTIMODULE boards include the iSBX 351 (one ASYNC/SYNC serial channel) and the iSBX 352 (one HDLC/SDLC serial channel) boards. Adding two iSBX 352 MULTIMODULE boards to the iSBC 88/45 ADCP provides a total of five HDLC/SDLC channels.

MULTIBUS® Multimaster Capabilities

OVERVIEW

The MULTIBUS system is Intel's industry standard microcomputer bus structure. Both 8- and 16-bit single board computers are supported on the MULTIBUS structure with 24 address and 16 data lines. In addition to expanding functions contained on a single board computer (e.g., memory and digital I/O), the MULTIBUS structure allows very powerful distributed processing configurations with multiple processors, intelligent slaves, and peripheral boards.

Multimaster Capability

The iSBC 88/45 ADCP board provides full MULTIBUS arbitration control logic. This control

Table 4. Interrupt Request Sources

Device	Function	No. of Interrupts
MULTIBUS® Interface	Select 1 interrupt from MULTIBUS® resident peripherals or other CPU boards	8
8273 HDLC/SDLC Controller	Transmit buffer empty and receive buffer full	2
8274 HDLC/SDLC SYNC/ASYNC Controller	Software examines register for status of communication operation	1
8254-Timer	Counter 2 of both PIT devices	2
iSBX™ Connectors	Function determined by iSBX™ MULTIMODULE™ Board (2 interrupts per socket)	4
Bus Fail Safe Timer	Indicates MULTIBUS® addressed device has not responded to command within 4 msec	1
Power Line Clock	Source of 60 MHz signal from power supply	1
Bus Flag Interrupt	Flag interrupt in byte location 1000H signals board reset or data handling request	2
iSBC® 337 Board	Numeric Data Processor generated status information	1
8237A-5	Signals end of 8237 DMA operation	1

logic allows up to three iSBC 88/45 ADCP boards or other bus masters, including iSBC 80 and iSBC 86 family boards to share the system bus using a serial (daisy chain) priority scheme. By using an external parallel priority decoder, the MULTIBUS system bus could be shared among sixteen masters.

The Intel standard MULTIBUS Interprocessor Protocol (MIP) software, implemented as the Intel iMMX 800 package for iRMX 80, iRMX 86, and iRMX 88 Real-Time Executives, fully supports multiple 8-and 16-bit distributed processor functions. The software manages the message passing protocol between microprocessors.

System Development Capabilities

The application development cycle for an iSBC 88/45 ADCP board is reduced and simplified through the usage of several Intel tools. The tools include the Intellec Series Microcomputer Development System, the ICE-88 In-Circuit Emulator, the iSBC 957B debug monitor software, and the iRMX 86 and iRMX 88 run-time support packages.

The Intellec Series Microcomputer Development System offers a complete development environment for the iSBC 88/45 software. In addition to the operating system, assembler, utilities and application debugger features provided with the system, the user optionally can utilize higher-level languages like PL/M, PASCAL, and FORTRAN.

The ICE-88 In-Circuit Emulator provides a link between the Intellec system and the target iSBC 88/45-based system for code loading and execution. The ICE-88 package assists the developer with the debugging and system integrating processes.

Run-Time Building Blocks

Intel offers run-time foundation software to support applications which range from general purpose to high-performance solutions. The iRMX 88 Real-time Multitasking Executive provides a multitasking structure which includes task scheduling, task management, intertask communications, and interrupt servicing for high-performance applications. The highly configurable modules make the system tailoring job easier whether one uses the compact executive or the complete executive with its variety of peripheral devices supported.

The iRMX 86 Operating System provides a very rich set of features and options to support sophisticated applications solutions. In addition to supporting real-time requirements, the iRMX 86 Operating System has a powerful, but easy-to-use human interface. When added to the sophisticated I/O system, the iRMX 86 Operating System is readily extended to support assembler, PL/M, PASCAL, and FORTRAN software development environments. The modular building block software lends itself well to customized application solutions.

SPECIFICATIONS

Word Size

Instruction — 8, 16, 24, or 32 bits

Data — 8 or 16 bits

System Clock

8 MHz — $\pm 0.1\%$

NOTE: Jumper selectable for 4 MHz operation with iSBC 337 Numeric Data Processor module or ICE-88 product.

Cycle Time

Basic Instruction Cycle at 8.00 MHz — 1.25 μ sec, 250 nsec (assumes instruction in the queue)

NOTE: Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles).

Memory Cycle Time

RAM — 500 nsec (no wait states)

EPROM — jumper selectable from 500 nsec to 625 nsec.

On-Board RAM* —

K Bytes	Hex Address Range
16 (total)	0000-3FFF
12 (dual-ported)	1000-3FFF

* Four iSBC 88/45 EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs (3 sockets); iSBC 341 (4 sockets)

Environmental Characteristics

Temperature — 0-55°C, free moving air across the base board and MULTIMODULE board

Humidity — 90%, non-condensing

Physical Characteristics

Width — 30.48 cm (12.00 in)

Length — 17.15 cm (6.75 in)

Height — 1.50 cm (0.59 in)

Weight — 6.20 gm (22 oz)

Memory Capacity/Addressing

On-Board EPROM* —

Device	Total K Bytes	Hex Address Range
2716	8	FE000-FFFF
2732A	16	FC000-FFFF
2764	32	F8000-FFFF
27128	64	F0000-FFFF

With optional

ISBC® 341 MULTIMODULE™ EPROM —

Device	Total K Bytes	Hex Address Range
2716	16	FC000-FFFF
2732A	32	F8000-FFFF
2764	64	F0000-FFFF
27128	128	E0000-FFFF

* Four ISBC 88/45 EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs (3 sockets); ISBC 341 sockets also support EPROMs and RAMs.

Timer Input Frequency — 8.00 MHz ± 0.1%

Interfaces

ISBX™ Bus — All signals TTL compatible

Serial RS232C Signals —

CTS	CLEAR TO SEND
DSR	DATA SET READY
DTE TXC	TRANSMIT CLOCK
DTR	DATA TERMINAL READY
FG	FRAME GROUND
RTS	REQUEST TO SEND
RXC	RECEIVE CLOCK
RXD	RECEIVE DATA
SG	SIGNAL GROUND
TXD	TRANSMIT DATA

Serial RS422A/449 Signals —

CS	CLEAR TO SEND
DM	DATA MODE
RC	RECEIVE COMMON
RD	RECEIVE DATA
RS	REQUEST TO SEND
RT	RECEIVE TIMING
SC	SEND COMMON
SD	SEND DATA
SG	SIGNAL GROUND
TR	TERMINAL READY
TT	TERMINAL TIMING

Electrical Characteristics

DC Power Dissipation — 28.3 Watts

DC Power Requirements —

Configuration	Current Requirements (all voltages ± 5%)		
	+ 5V	+ 12V	- 12V
without EPROM ¹	5.1A	20 mA	20 mA
with 8K EPROM (using 2716)	+ 0.14A	—	—
with 16K EPROM (using 2732A)	+ 0.20A	—	—
with 32K EPROM (using 2764)	+ 0.24A	—	—
with 64K EPROM (using 27128)	+ 0.24A	—	—

NOTE 1: AS SHIPPED — no EPROMs in sockets, no ISBC 341 module. Configuration includes terminators for two RS422A/449 and one RS232C channels.

Serial Communication Characteristics

Channel	Device	Supported Interface	Max. Baud Rate
A	8274 ¹	RS442A/449 RS232C CCITT V.24	800K SDLC/HDLC 125K Synchronous 50K Asynchronous
B	8274	RS232C CCITT V.24	125K Synchronous ² 50K Asynchronous
C	8273 ³	RS442A/449 RS232C CCITT V.24	64K SDLC/HDLC ³ 9.6K SELF CLOCKING

NOTES:

- 8274 supports HDLC/SDLC/SYNC/ASYN multiprotocol
- Exceed RS232C/CCITT V.24 rating of 20K baud
- 8273 supports HDLC/SDLC

BAUD RATE EXAMPLES (Hz)

8254 Timer Divide Count N	Synchronous K Baud	Asynchronous		
		÷ 16	÷ 32	÷ 64
		K Baud		
10	800	50.0	25.0	12.5
26	300	19.2	9.6	4.8
31	256	16.1	8.06	4.03
52	154	9.6	4.8	2.4
104	76.8	4.8	2.4	1.2
125	64	4.0	2.0	1.0
143	56	3.5	1.7	.87
167	48	3.0	1.5	.75
417	19.2	—	—	—
833	9.6	—	—	—
EQUATION	$\frac{8,000,000}{N}$	$\frac{500K}{N}$	$\frac{250K}{N}$	$\frac{125K}{N}$

SERIAL INTERFACE CONNECTORS

Interface	Mode ¹	MULTIMODULE™ Edge Connector	Cable	Connector
RS232C	DTE	26-pin ⁴ , 3M-3462-0001	3M ² -3349/25	25-pin ⁶ , 3M-3482-1000
RS232C	DCE	26-pin ⁴ , 3M-3462-0001	3M ² -3349/25	25-pin ⁶ , 3M-3483-1000
RS449	DTE	40-pin ⁵ , 3M-3464-0001	3M ³ -3349/37	37-pin ⁷ , 3M-3502-1000
RS449	DCE	40-pin ⁵ , 3M-3464-0001	3M ³ -3349/37	37-pin ⁷ , 3M-3503-1000

NOTES:

1. DTE — Data Terminal Equipment mode (male connector); DCE — Data Circuit Equipment mode (female connector) requires line swaps.
2. Cable is tapered at one end to fit the 3M-3462 connector.
3. Cable is tapered to fit 3M-3464 connector.
4. Pin 26 of the edge connector is not connected to the flat cable.
5. Pins 38, 39, and 40 of the edge connector are not connected to the flat cable.
6. May be used with the cable housing 3M-3485-1000.
7. Cable housing 3M-3485-4000 may be used with the connector.

Line Drivers (supplied)

Device	Characteristic	Qty	Installed
1488	RS232C	3	1
1489	RS232C	3	1
3486	RS422A	2	2
3487	RS422A	2	2

Reference Manual

143824 — iSBC 88/45 Advanced Data Communications Processor Board Hardware Reference Manual (not supplied).

Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051

ORDERING INFORMATION

Part Number	Description
SBC 88/45	8-bit 8088-based Single Board Computer with 3 HDLC/SDLC serial channels

Asynchronous +16 ÷ 32 ÷ 64 K Baud	Synchronous K Baud	Timer Divide Count N	8254
15.2	50.0	10	10
4.8	18.2	36	36
4.0	16.7	31	31
3.2	9.8	25	25
1.5	4.8	104	104
1.0	3.2	152	152
0.7	2.0	148	148
0.5	1.7	187	187
0.3	1.0	47	47
0.2	0.6	838	838
0.1	0.3	8000	8000
0.05	0.15	8000	8000
0.02	0.05	8000	8000
0.01	0.02	8000	8000



iSBC 88/40 MEASUREMENT AND CONTROL COMPUTER

- High performance 5 MHz iAPX 88/10 8-bit HMOS processor
- 12-bit, 20 kHz analog-to-digital converter with programmable gain control
- 16 differential/32 single-ended analog input channels
- Three iSBX MULTIMODULE connectors for analog, digital, and other I/O expansion
- 4K bytes static RAM, expandable via iSBC 301 MULTIMODULE RAM to 8K bytes (1K byte dual-ported)
- Four EPROM/E²PROM sockets for up to 64K bytes, expandable to 128K bytes with iSBC 341 expansion MULTIMODULE
- On-board 21-volt power supply for E²PROM modification under program control
- MULTIBUS Intelligent Slave or Multimaster

The Intel iSBC 88/40 Measurement and Control Computer is a member of Intel's large family of Single Board Computers that takes full advantage of Intel's VLSI technology to provide an economical self-contained computer based solution for applications in the areas of process control and data acquisition. The on-board iAPX 88/10 processor with its powerful instruction set allows users of the iSBC 88/40 board to update process loops as much as 5-10 times faster than previously possible with other 8-bit microprocessors. For example, the high performance iSBC 88/40 can concurrently process and update 16 control loops in less than 200 milliseconds using a traditional PID (Proportional-Integral-Derivative) control algorithm. The iSBC 88/40 board consists of a 16 differential/32 single ended channel analog multiplexer with input protected circuits, A/D converter, programmable central processing unit, dual port and private RAM, read only memory sockets, interrupt logic, 24 channels of parallel I/O, three programmable timers and MULTIBUS control logic on a single 6.75 by 12.00-inch printed circuit card. The iSBC 88/40 board is capable of functioning by itself in a stand-alone system or as a multimaster or intelligent slave in a large MULTIBUS system.

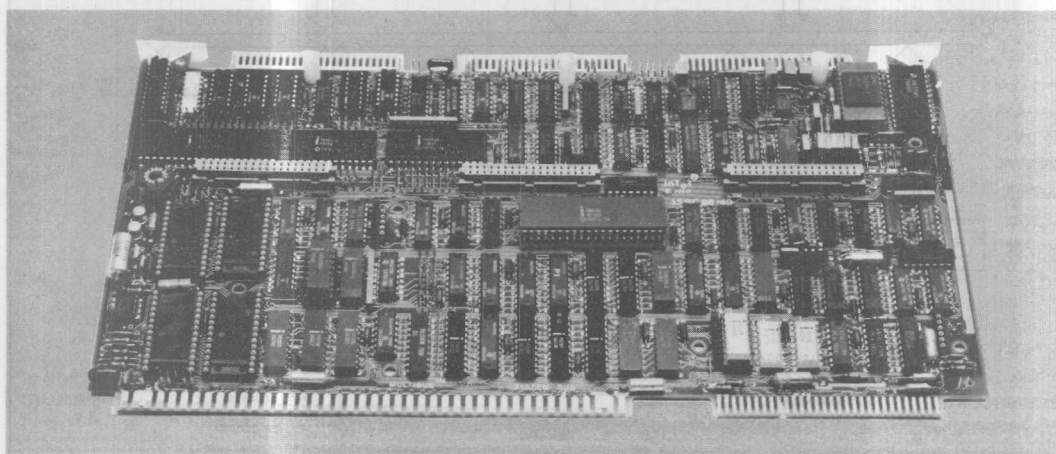


Figure 1. iSBC 88/40 Measurement and Control Computer Block Diagram

FUNCTIONAL DESCRIPTION

Three Modes of Operation

The iSBC 88/40 Measurement and Control Computer (MACC) is capable of operating in one of three modes: stand-alone controller, bus multimaster or intelligent slave. A block diagram of the iSBC 88/40 Measurement and Control Computer is shown in Figure 1.

Stand-Alone Controller

The iSBC 88/40 Measurement and Control Computer may function as a stand-alone single board controller with CPU, memory and I/O elements on a single board. The on-board 4K bytes of RAM and up to 64K bytes of read only memory, as well as the analog-to-digital converter and programable parallel I/O lines allow significant control and monitoring capabilities from a single board.

Bus Multimaster

In this mode of operation the iSBC 88/40 board may interface and control a wide variety of iSBC memory and I/O boards or even with additional

iSBC 88/40 boards or other single board computer masters or intelligent slaves.

Intelligent Slave

The iSBC 88/40 board can perform as an intelligent slave to any Intel 8 or 16-bit MULTIBUS master CPU by not only offloading the master of the analog data collection, but it can also do a significant amount of pre-processing and decision making on its own. The distribution of processing tasks to intelligent slaves frees the system master to do other system functions. The dual port RAM with flag bytes for signalling allows the iSBC 88/40 board to process and store data without MULTIBUS memory or bus contention.

Central Processing Unit

The central processor unit for the iSBC 88/40 board is a powerful 8-bit HMOS iAPX 88/10 micro-processor. The 22.5 sq. mil. chip contains approximately 29,000 transistors and has a clock rate of 5 MHz. The architecture includes four (4) addressable data registers and two (2) 16-bit memory base pointer registers and two (2) 16-bit index registers, all accessed by a total of 24 operand addressing modes for complex data handling and very flexible memory addressing.

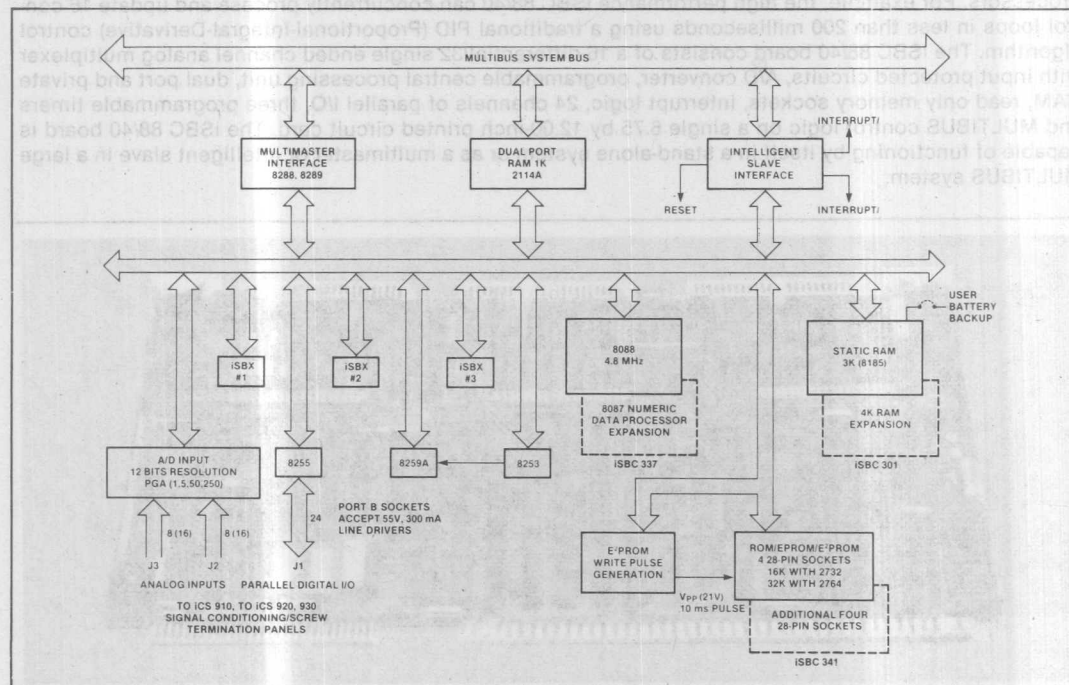


Figure 1. iSBC 88/40 Measurement and Control Computer Block Diagram

INSTRUCTION SET — The iAPX 88/10 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions. The instruction set of the iAPX 88/10 is a superset of the 8080A/8085A family and with available software tools, programs written for the 8080A/8085A can be easily converted and run on the iAPX 88/10 processor. Programs can also be run that are implemented on the iAPX 86/10 with little or no modification.

ARCHITECTURAL FEATURES — A 4-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 1.04 msec minimum instruction cycle to 417 nsec for queued instructions. The stack oriented architecture facilitates nested subroutines and co-routines, reentrant code and powerful interrupt handling. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions.

Bus Structure

The ISBC 88/40 single board computer has three buses: 1) an internal bus for communicating with on-board memory, analog-to-digital converter, ISBX MULTIMODULES and I/O options; 2) the MULTIBUS system bus for referencing additional memory and I/O options, and 3) the dual-port bus which allows access to RAM from the on-board CPU and the MULTIBUS system bus. Local (on-board) accesses do not require MULTIBUS communication, making the system bus available for use by other MULTIBUS masters (i.e. DMA devices and other single board computers transferring to additional system memory). This feature allows true parallel processing in a multiprocessor environment. In addition, the MULTIBUS interface can be used for system expansion through the use of other 8- and 16-bit ISBC computers, memory and I/O expansion boards.

RAM Capabilities

DUAL-PORT RAM — The dual-port RAM of the ISBC 88/40 board consists of 1K bytes of static

RAM, implemented with Intel 2114A chips. The on-board base address of this RAM is 00C00 (3K) normally; it is relocated to 01C00 (7K) when the ISBC 301 MULTIMODULE RAM is added to the protected RAM. The MULTIBUS port base address of the dual-port RAM can be jumpered to any 1K byte boundary in the 1M byte address space. The dual-port RAM can be accessed in a byte-wide fashion from the MULTIBUS system bus. When accessed from the MULTIBUS system bus, the dual-port RAM decode logic will generate INH1/ (Inhibit RAM) to allow dual-port RAM to overlay other system RAM. The dual-port control logic is designed to favor an on-board RAM access. If the dual-port is not currently performing a memory cycle for the MULTIBUS system port, only one wait state will be required. The on-board port may require more than one wait state if the dual-port RAM was busy when the on-board cycle was requested. The LOCK prefix facility of the iAPX 88/10 assembly language will disallow system bus accesses to the dual-port RAM. In addition, the on-board port to the dual-port RAM can be locked by other compatible MULTIBUS masters, which allows true symmetric semaphore operation. When the board is functioning in the master mode, the LOCK prefix will additionally disable other masters from obtaining the system bus.

PRIVATE RAM — In addition to the 1K byte dual-port RAM, there is a 3K byte section of private static RAM not accessible from the system bus. This RAM has a base address of 00000, and consists of three Intel 8185 RAM chips which are interfaced to the multiplexed address/data bus of the iAPX 88/10 microprocessor. Expansion of this private RAM from 3K to 7K bytes can be accomplished by the addition of an ISBC 301 MULTIMODULE RAM (4K bytes). When the 301 is added, protected RAM extends from 0 to 7K, and the base address of the dual-port RAM is relocated from 3K (00C00) to 7K (01C00). All protected RAM accesses require one wait state. The private RAM resides on the local on-board bus, which eliminates contention problems between on-board accesses to private RAM and system bus accesses to dual-port RAM. The private RAM can be battery backed (up to 16K bytes).

Additional RAM can be added by utilizing JEDEC-compatible static RAMs in the available EPROM sockets.

Parallel I/O Interface

The ISBC 88/40 single board computer contains 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral In-

terface. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports indicated in Table 1. There the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Port 2 can also accept TTL compatible peripheral drives, such as 75461/462, 75471/472, etc. These are open collector, high voltage drivers (up to 55 volts) which can sink 300 mA. Hence, the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector that mates with flat, woven, or round cable. This edge connector is also compatible with the Intel iCS 920 Digital I/O and iCS 930 AC Signal Conditioning/Termination Panels, for field wiring, optical isolation and high power (up to 3 amp) power drive.

EPROM Capabilities

Four (4) 28-pin sockets are provided for the use of Intel 2716s, 2732s, 2764s, 27128s, future JEDEC-compatible 128K and 256K bit EPROMs and their respective ROMs. When using 27128s the on-board EPROM capacity is 64K bytes. Read only memory expansion is available through the use of the iSBC 341 EPROM/ROM memory expansion MULTIMODULE. When the iSBC 341 is used an additional four (4) EPROM sockets are made available, for a total iSBC 88/40 board capacity of 128K bytes EPROM with Intel 27128s.

E²PROM Capabilities

The four 28-pin sockets can also accommodate Intel 2816 E²PROMs, for dynamic storage of control loop setpoints, conversion parameters, or other data (or programs) that change periodically but must be kept in nonvolatile storage. To give the user dynamic control of this nonvolatile memory, the iSBC 88/40 board also contains an on-board DC to DC converter which under program control will furnish the voltage necessary for modifying the contents of Intel 2816/2815 E²PROMs.

Timing Logic

The iSBC 88/40 board provides an 8253-5 Programmable Interval Timer, which contains three independent, programmable 16-bit timers/event counters. All three of these counters are available to generate time intervals or event counts under software control. The outputs of the three counters may be independently routed to the interrupt matrix. The inputs and outputs of timers 0 and 1 can be connected to parallel I/O lines on the J1 connector, where they replace 8255A port C lines. The third counter is also used for timing E²PROM write operations.

Interrupt Capability

The iSBC 88/40 board provides 9 vectored interrupt levels. The highest level is the NMI (Non-maskable Interrupt) line which is directly tied to the iAPX 88/10 CPU. This interrupt cannot be inhibited by software and is typically used for signalling catastrophic events (i.e., power failure). On servicing this interrupt, program control will be implicitly transferred through location 00008_H. The Intel 8259A Programmable Interrupt Controller (PIC) provides vectoring for the next eight interrupt levels. As shown in Table 2, a selection

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation				Control	
		Unidirectional					
		Input		Output			Bidirectional
		Latched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	4	X		X		X ¹	
	4	X		X		X ¹	

NOTE:

1. Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

of four priority processing modes is available to the designer to match system requirements. Operating mode and priority assignments may be re-configured dynamically via software at any time during system operation. The PIC accepts interrupt requests from the programmable parallel and/or ISBX interfaces, the programmable timers, the system bus, or directly from peripheral equipment. The PIC then determines which of the incoming requests is of the highest priority, determines whether this request is of higher priority than the level currently being serviced, and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. The PIC generates a unique memory address for each interrupt level. These addresses are equally spaced at 4-byte intervals. This 32-byte block may begin at any 32-byte boundary in the lowest 1K bytes of memory*, and contains unique instruction pointers and code segment offset values (for expanded memory operation) for each interrupt level. After acknowledging an interrupt and obtaining a device identifier byte from the 8259A PIC, the CPU will store its status flags on the stack and execute an indirect CALL instruction through the vector location (derived from the device identifier) to the interrupt service routine.

*NOTE: The first 32 vector locations are reserved by Intel for dedicated vectors. Users who wish to maintain compatibility with present and future Intel products should not use these locations for user-defined vector addresses.

Table 2. Programmable Interrupt Modes

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

INTERRUPT REQUEST GENERATION — Interrupt requests may originate from 26 sources. Two jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface when a byte of information is ready to be transferred to the CPU (i.e., input buffer is

full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). A jumper selectable request can be generated by each of the programmable timers. An additional interrupt request line may be jumpered directly from the parallel I/O driver terminator section. Eight prioritized interrupt request lines allow the ISBC 88/40 board to recognize and service interrupts originating from peripheral boards interfaced via the MULTIBUS system bus. The fail safe timer can be selected as an interrupt source. Also, interrupts are provided from the ISBX connectors (6), end-of-conversion, PFIN and from the power line clock.

Power-Fail Control

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the ISBC 635, ISBC 640, and ICS 645 Power Supply or equivalent.

ISBX MULTIMODULE Expansion Capabilities

Three ISBX MULTIMODULE connectors are provided on the ISBC 88/40 board. Up to three (3) single wide MULTIMODULE or one (1) double wide and one (1) single wide ISBX MULTIMODULE can be added to the ISBC 88/40 board. A wide variety of peripheral controllers, analog and digital expansion options are available. For more information on specific ISBX MULTIMODULES consult the Intel OEM Microcomputer System Configuration Guide.

Processing Expansion Capabilities

The addition of a ISBC 337 Multimodule Numeric Data Processor offers high performance integer and floating point math functions to users of the ISBC 88/40 board. The ISBC 337 incorporates the Intel 8087 and because of the MULTIMODULE implementation, it allows on-board expansion directly on the ISBC 88/40 board, eliminating the need for additional boards for floating point requirements.

MULTIBUS Expansion

Memory and I/O capacity may be expanded further and additional functions added using Intel MULTIBUS compatible expansion boards. Memory may be expanded by adding user specified combinations of RAM boards, EPROM boards, or memory combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O MULTIBUS expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers, or hard disk con-

trollers either through the use of expansion boards and ISBX MULTIMODULES. Modular expandable backplanes and cardcages are available to support multiboard systems.

NOTE: Certain system restrictions may be incurred by the inclusion of some of the ISBC 80 family options in an ISBC 88/40 system. Consult the Intel OEM Microcomputer System Configuration Guide for specific data.

Analog Input Section

The analog section of the ISBC 88/40 board receives all control signals through the local bus to initiate channel selection, gain selection, sample and hold operation, and analog-to-digital conversion. See Figure 2.

INPUT CAPACITY — 32 separate analog signals may be randomly or sequentially sampled in single-ended mode with the 32 input multiplexers and a common ground. For noisier environments, differential input mode can be configured to achieve 16 separate differential signal inputs, or 32 pseudo differential inputs.

RESOLUTION — The analog section provides 12-bit resolution with a successive approximation analog-to-digital converter. For bipolar operation (-5 to $+5$ or -10 to $+10$ volts) it provides 11 bits plus sign.

SPEED — The A-to-D converter conversion speed is $50 \mu\text{s}$ (20 kHz samples per second). Combined with the programming interface, maximum throughput via the local bus and into memory will be 55 microseconds per sample, or 18 kHz samples per second, for a single channel, a random channel, or a sequential channel scan at a gain of

1, 5 ms at a gain of 5, 250 ms at a gain of 50, and 20 ms at a gain of 250. A-to-D conversion is initiated via a programmed command from the iAPX 88/10 central processor. Interrupt on end-of-conversion is a standard feature to ease programming and timing constraints.

ACCURACY — High quality components are used to achieve 12 bits resolution and accuracy of 0.035% full scale range $\pm \frac{1}{2}$ LSB. Offset is adjustable under program control to obtain a nominal $\pm 0.024\%$ FSR $\pm \frac{1}{2}$ LSB accuracy at any fixed temperature between 0°C and 60°C (gain = 1). See specifications for other gain accuracies.

GAIN — To allow sampling of millivolt level signals such as strain gauges and thermocouples, gain is made configurable via user program commands up to $250 \times$ (20 millivolts full scale input range). User can select gain ranges of 1 (5V), 5 (1V), 50 (100 mV), 250 (20 mV) to match his application.

OPERATIONAL DESCRIPTION — The ISBC 88/40 single board computer addresses the analog-to-digital converter by executing IN or OUT instructions to the port address. Analog-to-digital conversions can be programmed in either of two modes: 1) start conversion and poll for end-of-conversion (EOC), or 2) start conversion and wait for interrupt at end of conversion. When the conversion is complete as signaled by one of the above techniques, INput instructions read two bytes (low and high bytes) containing the 12-bit data word as shown on the following page.

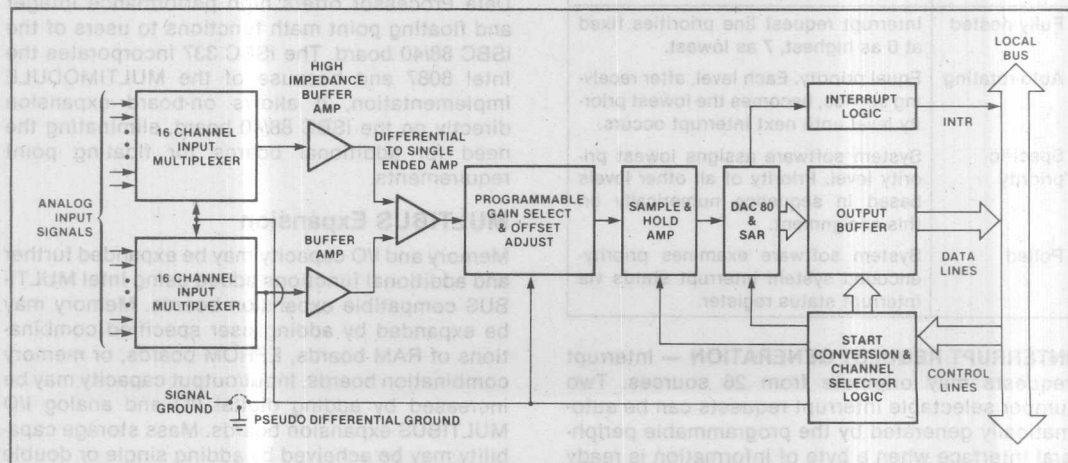


Figure 2. ISBC 88/40 Analog Input Section

Output Command — Select input channel and start conversion.

BIT POSITION	GAIN		CONNECTOR		CHANNEL SELECT			
	7	6	5	4	3	2	1	0
INPUT CHANNEL	G1	G2	J	C3	C2	C1	C0	

Input Data — Read converted data (low byte) or Read converted data (high byte).

BIT POSITION	DATA							
	7	6	5	4	3	2	1	0
LOW/STATUS BYTE	D3	D2	D1	D0				EOC

BIT POSITION	DATA							
	7	6	5	4	3	2	1	0
HIGH BYTE	D11	D10	D9	D8	D7	D6	D5	D4

Offset Correction — At higher gains ($\times 50$, $\times 250$) the voltage offset tempco in the A/D circuitry can sometimes cause unacceptable inaccuracies. To correct for this offset, one channel can be dedicated to be used as a reference standard. This channel can be read from the program to deter-

mine the amount of offset. The reading from this channel will then be subtracted from all other channel readings, in effect eliminating the offset tempco.

System Software Development

The development cycle of the iSBC 88/40 board may be significantly reduced using an Intel Inteltec Microcomputer Development System with the optional iAPX 88/iAPX 86 Software Development package.

The iAPX 88/iAPX 86 Software Development package includes Intel's high-level programming language, PL/M 86. PL/M 86 provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M 86 programs can be written in a much shorter time than assembly language programs for a given application.

SPECIFICATIONS

Word Size

Instruction — 8, 16, or 32 bits

Data — 8 bits

Instruction Cycle Time

417 nanoseconds for fastest executable instruction (assumes instruction is in the queue). 1.04 microseconds for fastest executable instruction (assumes instruction is not in the queue).

Memory Capacity

On-board ROM/EPROM/E²PROM

Up to 64K bytes; user installed in 2K, 4K, 8K or 16K byte increments or up to 128K if iSBC 341 MULTIMODULE EPROM option installed. Up to 8K bytes of E²PROM using Intel 2816s may be user-installed in increments of 2, 4 or 8K bytes.

On-board RAM

4K bytes or 8K bytes if the iSBC 301 MULTIMODULE RAM is installed. Integrity maintained during power failure with user-furnished batteries. 1K bytes are dual-ported.

Off-board Expansion

Up to 1 megabyte of user-specified combination of RAM, ROM, and EPROM.

Memory Addressing

On-board ROM/EPROM

FE000-FFFF (using 2716 EPROMs)
FC000-FFFF (using 2732 EPROMs)
F8000-FFFF (using 2764 EPROMs)
F0000-FFFF (using 27128 EPROMs)

On-board ROM/EPROM (With iSBC 341 MULTIMODULE EPROM option installed)

FC000-FFFF (using 2716 EPROMs)
F8000-FFFF (using 2732 EPROMs)
F0000-FFFF (using 2764 EPROMs)
E0000-FFFF (using 27128 EPROMs)

On-board RAM (CPU Access)

00000-00FFF
00000-01FFF (if iSBC 301 MULTIMODULE RAM option installed)

On-board RAM

Jumpers allow 1K bytes of RAM to act as slave RAM for access by another bus master. Addressing may be set within any 1K boundary in the 1-megabyte system address space.

Slave RAM Access

Average; 350 nanoseconds

Interval Timer

Output Frequencies —

Function	Single Timer		Dual Timers (Two Timers Cascaded)
	Min.	Max.	
Real-Time Interrupt Interval	0.977 μ s	64 ms	69.9 minutes maximum
Rate Generator (Frequency)	15.625 Hz	1024 kHz	0.00024 Hz minimum

iAPX 88/10 CPU Clock

4.8 MHz $\pm 0.1\%$

I/O Addressing

All communications to parallel I/O ports, iSBX bus, A/D port, timers, and interrupt controller are via read and write commands from the on-board iAPX 88/10 CPU.

Interface Compatibility

Parallel I/O — 24 programmable lines (8 lines per port); one port includes a bidirectional bus driver. IC sockets are included for user installation of line drivers and/or I/O terminators and/or peripheral drivers as required for interface ports.

iSBX Bus Connectors — Three iSBX bus connectors are provided. These connectors accept 8-bit iSBX MULTIMODULE boards. One set of the three iSBX MULTIMODULE connectors will accept a double wide iSBX MULTIMODULE board.

Interrupts

iAPX 88/10 CPU includes a non-maskable interrupt (NMI). NMI interrupt is provided for catastrophic events such as power failure. The on-board 8259A PIC provides 8-bit identifier of interrupting device to CPU. CPU multiplies identifier by four to derive vector address. Jumpers select interrupts from 26 sources without necessity of external hardware. PIC may be programmed to accommodate edge-sensitive or level-sensitive inputs.

Analog Input

16 differential (bipolar operation) or 32 single-ended (unipolar operation).

Full Scale Voltage Range — -5 to +5 volts (bipolar), 0 to +5 volts (unipolar).

NOTE: Ranges of 0 to 10V and $\pm 10V$ achievable with externally supplied $\pm 15V$ power.

Gain — Program selectable for gain of 1, 5, 50, or 250.

Resolution — 12 bits (11 bits plus sign for ± 5 , ± 10 volts).

Accuracy — Including noise and dynamic errors

Gain	25°C
1	$\pm 0.05\%$ FSR*
5	$\pm 0.075\%$ FSR*
50	$\pm 0.085\%$ FSR*
250	$\pm 0.12\%$ FSR*

* **NOTE:** FSR = Full Scale Range $\pm \frac{1}{2}$ LSB. Figures are in percent of full scale reading. At any fixed temperature between 0°C and 60°C, the accuracy is adjustable to $\pm 0.05\%$ of full scale.

Gain TC (at gain = 1) — 30 PPM (typical), 56 PPM (max) per degree centigrade, 40 PPM at other gains.

Offset TC — (in % of FSR/°C)	Gain	Offset TC (typical)
	1	0.0018%
	5	0.0036%
	50	0.024%
	250	0.12%

Sample and Hold-sample Time — 15 μs

Aperture-hold Aperture Time — 120 ns

Input Overvoltage Protection — 30 volts

Input Impedance — 20 megohms (min.)

Conversion Speed — 50 μs (max.) at gain = 1

Common Mode Rejection Ratio — 60 dB (min.)

Physical Characteristics

Width — 30.48 cm (12.00 in.)

Length — 17.15 cm (6.75 in.)

Height — 1.78 cm (0.7 in.)

2.82 cm (1.13 in.) with iSBC Memory Expansion, MULTIMODULES, iSBX Numeric Data Processor or iSBX MULTIMODULES.

Electrical Requirements

Power Requirements (Maximum) —

Config- uration	+5V		+5V Aux		+12V		-12V	
	Typ	Max	Typ	Max	Typ	Max	Typ	Max
iSBC 88/40 ^{1,2}	4	5.5	100	150	80	120	30	40

NOTES:

- The current requirement includes one worst case (active-standby) EPROM current.
- If +5V Aux is supplied by the iSBC 88/40 board, the total +5V current is the sum of the +5V and the +5V Aux.

Environmental Requirements

Operating Temperature — 0° to 55°C (32°C to 131°F) with 200 lfm air flow

Relative Humidity — to 90% without condensation

Equipment Supplied

The following are supplied with the iSBC 88/40 board:

- Schematic diagram
- Assembly drawing

Reference Manuals

124978-001 — iSBC 88/40 Measurement and Control Computer Hardware Reference Manual (NOT SUPPLIED).

Manuals may be ordered from an Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

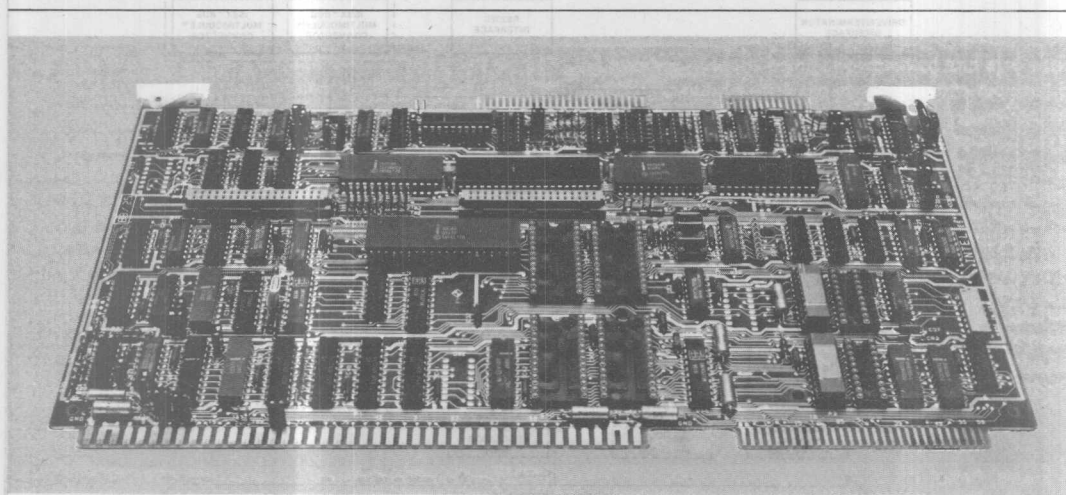
Part Number Description

SBC 88/40	Measurement and Control Computer
-----------	----------------------------------

iSBC™ 88/25 SINGLE BOARD COMPUTER

- 8-bit 8088 Microprocessor operating at 5 MHz
- One megabyte addressing range
- Two iSBX™ bus connectors
- Optional Numeric Data Processor with iSBC 337 MULTIMODULE™ Processor
- 4K bytes of static RAM; expandable on-board to 16K bytes
- Sockets for up to 64K bytes of JEDEC 24/28-pin standard memory devices; expandable on-board to 128K bytes
- Programmable synchronous/asynchronous RS232C compatible serial interface with software selectable baud rates
- 24 programmable parallel I/O lines
- Two programmable 16-bit BCD or binary timers/event counters
- 9 Levels of vectored interrupt control, expandable to 65 levels
- MULTIBUS® interface for multimaster configurations and system expansion
- Development support with Intel's iPDS, low cost Personal Development System, and EMV-88 Emulator

The iSBC 88/25 Single Board Computer is a member of Intel's complete line of OEM microcomputer systems which take full advantage of Intel's technology to provide economical, self-contained, computer-based solutions for OEM applications. The iSBC 88/25 board is a complete computer system on a single 6.75 x 12.00-in. printed circuit card. The CPU, system clock, read/write memory, nonvolatile read only memory, I/O ports and drivers, serial communications interface, priority interrupt logic and programmable timers, all reside on the board. The large control storage capacity makes the iSBC 88/25 board ideally suited for control-oriented applications such as process control, instrumentation, industrial automation, and many others.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: CREDIT, Index, Intel, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, uScope, Promware, MCS, ICE, IRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

FUNCTIONAL DESCRIPTION

Central Processing Unit

The central processor for the iSBC 88/25 board is Intel's 8088 CPU operating at 5 MHz. The CPU architecture includes four 16-bit byte addressable data registers, two 16-bit memory base pointer registers and two 16-bit index registers, all accessed by a total of 24 operand addressing modes for comprehensive memory addressing and for support of the data structures required for today's structured, high level languages, as well as assembly language.

Instruction Set

The 8088 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions.

For enhanced numerics processing capability, the iSBC 337 MULTIMODULE Numeric Data Processor extends the architecture and data set. Over 60 numeric instructions offer arithmetic, trigonometric, transcendental, logarithmic and exponen-

tial instructions. Supported data types include 16, 32, and 64-bit integer, and 32 and 64-bit floating point, 18-digit packed BCD and 80-bit temporary.

Architectural Features

A 4-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 750 nsec minimum instruction cycle to 250 nsec for queued instructions. The stack-oriented architecture readily supports modular programming by facilitating fast, simple, inter-module communication, and other programming constructs needed for asynchronous real-time systems. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions. All Intel® languages support the extended memory capability, relieving the programmer of managing the megabyte memory space, yet allowing explicit control when necessary.

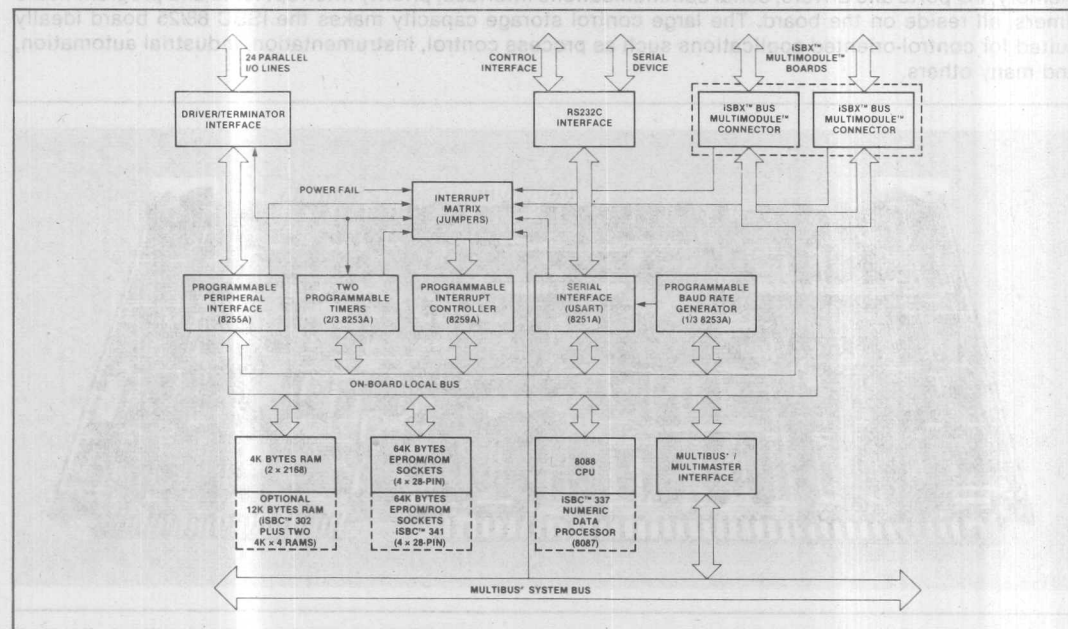


Figure 1. iSBC 88/25 Block Diagram

Memory Configuration

The iSBC 88/25 microcomputer contains 4K bytes of high-speed static RAM on-board. In addition, the on-board RAM may be expanded to 12K bytes via the iSBC 302 8K byte RAM module which mounts on the iSBC 88/25 board and then to 16K bytes by adding two 4K×4 RAM devices in sockets on the iSBC 302 module. All on-board RAM is accessed by the 8088 CPU with no wait states, yielding a memory cycle time of 800 nsec.

In addition to the on-board RAM, the iSBC 88/25 board has four 28-pin sockets, configured to accept JEDEC 24/28-pin standard memory devices. Up to 64K bytes of EPROM are supported in 16K-byte increments with Intel 27128 EPROMs. The iSBC 88/25 board is also compatible with the 2716, 2732, and 2764 EPROMs allowing a capacity of 8K, 16K, and 32K bytes, respectively. Other JEDEC standard pinout devices are also supported, including byte-wide static and integrated RAMs.

With the addition of the iSBC 341 MULTIMODULE EPROM option, the on-board capacity for these devices is doubled, providing up to 128K bytes of EPROM capacity on-board.

Parallel I/O Interface

The iSBC 88/25 Single Board Computer contains 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral Interface. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports indicated in Table 1. In order to take advantage of the

large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators, allowing the selection of the appropriate combination of optional line drivers and terminators with the required drive/termination characteristics. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector.

Serial I/O

A programmable communications interface using the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 88/25 board. A software selectable baud rate generator provides the USART with all common communication frequencies. The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The RS232C compatible interface on each board, in conjunction with the USART, provides a direct interface to RS232C compatible terminals, cassettes, and asynchronous and synchronous modems. The RS232C command lines, serial data lines and signal ground line are brought out to a 26-pin edge connector.

Programmable Timers

The iSBC 88/25 board provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation				Control	
		Unidirectional					
		Input		Output			Bidirectional
		Latched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	4	X		X		X ¹	
	4	X		X		X ¹	

NOTE:

1. Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller and to the I/O terminators associated with the 8255A to allow external devices or an 8255A port to gate the timer or to count external events. The third interval timer in the 8253 provides the programmable baud rate generator for the iSBC 88/25 board RS232C USART serial port. The system software configures each timer independently to select the desired function. Seven functions are available as shown in Table 2. The contents of each counter may be read at any time during system operation.

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counter "window" has been enabled or an interrupt may be generated after N events occur in the system.

iSBX MULTIMODULE On-Board Expansion

Two 8-bit iSBX MULTIMODULE connectors are provided on the iSBC 88/25 microcomputer. Through these connectors, additional on-board I/O functions may be added. iSBX MULTIMODULES optimally support functions provided by VLSI peripheral components such as additional parallel and serial I/O, analog I/O, small mass storage device controllers (e.g., cassettes and floppy disks), and other custom interfaces to meet specific needs. By mounting directly on the single board computer, less interface logic, less power, simpler packaging, higher performance, and lower cost result when compared to other alternatives such as MULTIBUS form factor compatible boards. The iSBX connectors on the iSBC 88/25 provide all signals necessary to interface to the local on-board bus. A broad range of iSBX MULTIMODULE options are available in this family from Intel. Custom iSBX modules may also be designed for use on the iSBC 88/25 board. An iSBX bus interface specification and iSBX connectors are available from Intel.

MULTIBUS SYSTEM BUS AND MULTIMASTER CAPABILITIES

Overview

The MULTIBUS system bus is Intel's industry standard microcomputer bus structure. Both 8 and 16-bit single board computers are supported on the MULTIBUS structure with 24 address and 16 data lines. In its simplest application, the MULTIBUS system bus allows expansion of functions already contained on a single board computer (e.g., memory and digital I/O). However, the MULTIBUS structure also allows very powerful distributed processing configurations with multiple processors and intelligent slave I/O, and peripheral boards capable of solving the most demanding microcomputer applications. The MULTIBUS system bus is supported with a broad array of board level products, LSI interface components, detailed published specifications and application notes.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. Memory may be expanded by adding user specified combinations

of RAM boards, EPROM boards, or combination boards. Input/output capacity may be added with digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers, or hard disk controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

Multimaster Capabilities

For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication of the system bus), the iSBC 88/25 board provides full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 88/25 boards or other bus masters, including iSBC 80 and iSBC 86 family MULTIBUS compatible single board computers to share the system bus using a serial (daisy chain) priority scheme and allows up to 16 masters to share the MULTIBUS system bus with an external parallel priority decoder. In addition to the multiprocessing configurations made possible with multimaster capability, it also provides a very efficient mechanism for all forms of DMA (Direct Memory Access) transfers.

Interrupt Capability

The iSBC 88/25 board provides 9 vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line which is directly tied to the 8088 CPU. This interrupt is typically used for signaling catastrophic events (e.g., power failure). The Intel 8259A Programmable Interrupt Controller (PIC) provides control and vectoring for the next eight interrupt levels. As shown in Table 3, a selection of four priority processing modes is available for use in designing request processing configurations to match system requirements for efficient interrupt servicing with minimal latencies. Operating mode and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from all on-board I/O resources and from the MULTIBUS system bus. The PIC then resolves requests according to the selected mode and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked via software, by storing a single byte in the interrupt mask register of the PIC. In systems requiring additional interrupt levels, slave 8259A PICs may be interfaced via the MULTIBUS system bus, to generate additional

vector addresses, yielding a total of 65 unique interrupt levels.

Table 3. Programmable Interrupt Modes

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

Interrupt Request Generation

Interrupt requests to be serviced by the iSBC 88/25 board may originate from 24 sources. Table 4 includes a list of devices and functions supported by interrupts. All interrupt signals are brought to the interrupt jumper matrix where any combination of interrupt sources may be strapped to the desired interrupt request level on the 8259A PIC or the NMI input to the CPU directly.

Power-Fail Control and Auxiliary Power

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the iSBC 635 and iSBC 640 Power Supply or equivalent, to initiate an orderly shut down of the system in the event of a power failure. Additionally, an active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power-down sequences. An auxiliary power bus is also provided to allow separate power to RAM for systems requiring battery backup of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

System Development Capabilities

The development cycle of iSBC 88/25 products can be significantly reduced and simplified by using the Intellec Series Microcomputer Development Systems. The Assembler, Locating Linker, Library Manager, Text Editor and System Monitor

are all supported by the ISIS-II disk-based operating system. To facilitate conversion of 8080A/8085A assembly language programs to run on the iSBC 88/25 board, CONV-86 is available under the ISIS-II operating system. The iSBC 88/25 board is also supported by Intel's iPDS, Personal Development System. This system provides low cost development support while at the same time providing personal computer capability for the engineer.

IN-CIRCUIT EMULATOR

The ICE-88 In-Circuit Emulator provides the necessary link between the software development environment provided by the Intellec system and the "target" iSBC 88/25 execution system. In addition to providing the mechanism for loading executable code and data into the iSBC 88/25 board, the ICE-88 In-Circuit Emulator provides a sophisticated command set to assist in debugging software and final integration of the user hardware and software. The EMV-88 Emulator, designed for 8088-based product support on the iPDS, provides for a complete development solution at low cost.

PL/M-86

Intel's system's implementation language, PL/M-86, is also available as an Intellec Microcom-

puter Development System option. PL/M-86 provides the capability to program in algorithmic language and eliminates the need to manage register usage or allocate memory while still allowing explicit control of the system's resources when needed.

Run-Time Support

Intel also offers two run-time support packages; iRMX 88 Realtime Multitasking Executive and the iRMX 86 Operating System. iRMX 88 is a simple, highly configurable and efficient foundation for small, high performance applications. Its multitasking structure establishes a solid foundation for modular system design and provides task scheduling and management, intertask communication and synchronization, and interrupt servicing for a variety of peripheral devices. Other configurable options include terminal handlers, disk file system, debuggers and other utilities. iRMX 86 is a high functional operating system with a very rich set of features and options based on an object-oriented architecture. In addition to being modular and configurable, functions beyond the nucleus include a sophisticated file management and I/O system, and powerful human interface. Both packages are easily customized and extended by the user to match unique requirements.

Table 4. Interrupt Request Sources

Device	Function	Number of Interrupts
MULTIBUS interface	Requests from MULTIBUS resident peripherals or other CPU boards	8; may be expanded to 64 with slave 8259A PIC's on MULTIBUS boards
8255A Programmable Peripheral Interface	Signals input buffer full or output buffer empty; also BUS INTR OUT general purpose interrupt from driver/terminator sockets	3
8251A USART	Transmit buffer empty and receive buffer full	2
8253 Timers	Timer 0, 1 outputs; function determined by timer mode	2
iSBX connectors	Function determined by iSBX MULTIMODULE board	4 (2 per iSBX connector)
Bus fail safe timer	Indicates addressed MULTIBUS resident device has not responded to command within 6 msec	1
Power fail interrupt	Indicates AC power is not within tolerance	1
Power line clock	Source of 120 Hz signal from power supply	1
External interrupt	General purpose interrupt from parallel port J1 connector	1
iSBC 337 MULTIMODULE Numeric Data Processor	Indicates error or exception condition	1

SPECIFICATIONS

Word Size

INSTRUCTION — 8, 16, 24, or 32 bits

DATA — 8 bits

System Clock

5.00 MHz or 4.17 MHz $\pm 0.1\%$ (jumper selectable)

NOTE: 4.17 MHz required with the optional iSBC 337 module.

Cycle Time

BASIC INSTRUCTION CYCLE

At 5 MHz — 1.2 μ sec

— 400 nsec (assumes instruction in the queue)

NOTES: Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles).

Memory Cycle Time

RAM — 800 nsec (no wait states)

EPROM — Jumper selectable from 800 nsec to 1400 nsec

Memory Capacity/Addressing

ON-BOARD EPROM

Device	Total Capacity	Address Range
2716	8K bytes	FE000-FFFFFH
2732	16K bytes	FC000-FFFFFH
2764	32K bytes	F8000-FFFFFH
27128	64K bytes	F0000-FFFFFH

WITH iSBC 341 MULTIMODULE EPROM

Device	Total Capacity	Address Range
2716	16K bytes	FC000-FFFFFH
2732	32K bytes	F8000-FFFFFH
2764	64K bytes	F0000-FFFFFH
27128	128K bytes	E0000-FFFFFH

NOTES: iSBC 88/25 EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs (2 sockets); iSBC 341 sockets also support E²PROMs.

ON-BOARD RAM

4K bytes — 0-0FFFH

WITH iSBC 302 MULTIMODULE RAM

12K bytes — 0-2FFFH

WITH iSBC 302 MULTIMODULE BOARD AND TWO 4K \times 4 RAM CHIPS

16K bytes — 0-3FFFH

I/O Capacity

PARALLEL — 24 programmable lines using one 8255A

SERIAL — 1 programmable line using one 8251A

iSBX MULTIMODULE — 2 iSBX MULTIMODULE boards

Serial Communications Characteristics

SYNCHRONOUS — 5-8 bit characters; internal or external character synchronization; automatic sync insertion

ASYNCHRONOUS — 5-8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection

BAUD RATES

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)	
	Synchronous	Asynchronous
153.6	—	+ 16 + 64 9600 2400
76.8	—	4800 1200
38.4	38400	2400 600
19.2	19200	1200 300
9.6	9600	600 150
4.8	4800	300 75
2.4	2400	150 —
1.76	1760	110 —

NOTES:

Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register (8253 Timer 2).

Timers

INPUT FREQUENCIES

Reference: 2.458 MHz $\pm 0.1\%$ (406.9 nsec period, nominal); or 1.229 MHz $\pm 0.1\%$ (813.8 nsec period, nominal); or 153.6 kHz $\pm 0.1\%$ (6.510 μ sec period, nominal)

NOTES:

Above frequencies are user selectable.

Event Rate: 2.46 MHz max

OUTPUT FREQUENCIES/TIMING INTERVALS

Function	Single Timer/Counter		Dual Timer/Counter (Two Timers Cascaded)	
	Min	Max	Min	Max
Real-time Interrupt	1.63 μ s	427.1 ms	3.26s	466.50 min
Programmable one-shot	1.63 μ s	427.1 ms	3.26s	466.50 min
Rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Square-wave rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Software triggered strobe	1.63 μ s	427.1 ms	3.26s	466.50 min
Hardware triggered strobe	1.63 μ s	427.1 ms	3.26s	466.50 min
Event counter	—	2.46 MHz	—	—

Interfaces

MULTIBUS — All signals TTL compatible

iSBX BUS — All signals TTL compatible

PARALLEL I/O — All signals TTL compatible

SERIAL I/O — RS232C compatible, configurable as a data set or data terminal

TIMER — All signals TTL compatible

INTERRUPT REQUESTS — All TTL compatible

Connectors

Interface	Double-Sided Pins (qty)	Centers (in.)	Mating Connectors
MULTIBUS System	86	0.156	Viking 3KH43/9AMK12 Wire Wrap
iSBX Bus 8-Bit Data	36	0.1	iSBX 960-5
Parallel I/O (2)	50	0.1	3M 3415-000 Flat or TI H312125 Pins
Serial I/O	26	0.1	3M 3462-0001 Flat or AMP 88106-1 Flat

Line Drivers and Terminators

I/O DRIVERS — The following line drivers are all compatible with the I/O driver sockets on the iSBC 88/25 board

Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

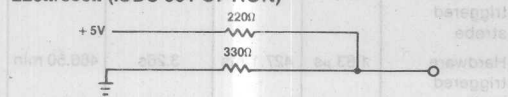
NOTES:

I = inverting; NI = non-inverting; OC = open collector.

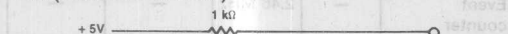
Port 1 of the 8255A has 32 mA totem-pole bidirectional drivers and 10 k Ω terminators

I/O TERMINATORS — 220 Ω /330 Ω divider or 1 k Ω pullup

220 Ω /330 Ω (iSBC 901 OPTION)



1 k Ω (iSBC 902 OPTION)



MULTIBUS Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-State	32
Address	Tri-State	24
Commands	Tri-State	32
Bus Control	Open Collector	20

Physical Characteristics

WIDTH — 12.00 in. (30.48 cm)

HEIGHT — 6.75 in. (17.15 cm)

DEPTH — 0.70 in. (1.78 cm)

WEIGHT — 14 oz (388 gm)

Electrical Characteristics

DC POWER REQUIREMENTS

Configuration	Current Requirements (All Voltages $\pm 5\%$)		
	+ 5V	+ 12V	- 12V
Without EPROM ¹	3.8A	25 mA	23 mA
RAM only ²	104 mA		
With 8K EPROM ³ (using 2716)	4.3A	25 mA	23 mA
With 16K EPROM ³ (using 2732)	4.4A	25 mA	23 mA
With 32K EPROM ³ (using 2764)	4.4A	25 mA	23 mA

NOTES:

- Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators.
- RAM chips powered via auxiliary power bus in power-down mode. Does not include power for optional RAM.
- Includes power required for 4 ROM/EPROM chips, and I/O terminators installed for 16 I/O lines; all terminator inputs low.

Environmental Characteristics

OPERATING TEMPERATURE — 0°C to 55°C

RELATIVE HUMIDITY — to 90% (without condensation)

Reference Manual

143825-001 — iSBC 88/25 Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

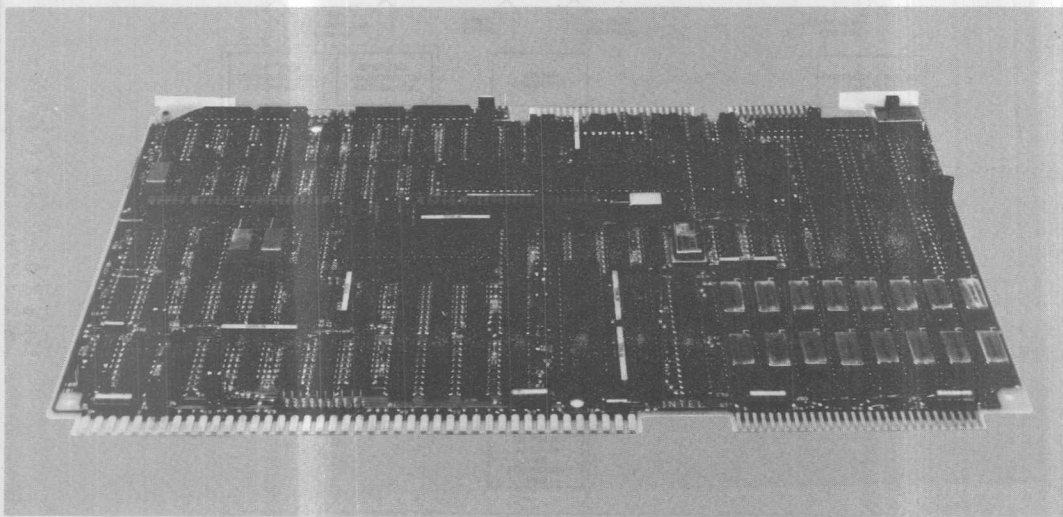
SBC 88/25 8-bit Single Board Computer with 4K bytes RAM



iSBC™ 86/14 and iSBC™ 86/30 SINGLE BOARD COMPUTERS

- iAPX 86/10 (8086-2) Microprocessor with 5 or 8 MHz CPU clock
- Fully software compatible with iSBC™ 86/12A Single Board Computer
- Optional iAPX 86/20 Numeric Data Processor with iSBC™ 337 MULTIMODULE™ processor
- 32K/128K bytes of dual-port read/write memory expandable on-board to 256K bytes with on-board refresh
- Sockets for up to 64K bytes of JEDEC 24/28-pin standard memory devices
- Two iSBX™ bus connectors
- 24 programmable parallel I/O lines
- Programmable synchronous/asynchronous RS232C compatible serial interface with software selectable baud rates
- Two programmable 16-bit BCD or binary timers/event counters
- 9 Levels of vectored interrupt control, expandable to 65 levels
- MULTIBUS® interface for multimaster configurations and system expansion
- Supported by a complete family of single board computers, memory, digital and analog I/O, peripheral controllers, packaging and software

The iSBC 86/14 and iSBC 86/30 Single Board Computers are members of Intel's complete line of OEM microcomputer systems which take full advantage of Intel's technology to provide economical, self-contained, computer-based solutions for OEM applications. Each board is a complete computer system on a single 6.75 x 12.00-in. printed circuit card distinguished by RAM memory content with 32K bytes and 128K bytes provided on the iSBC 86/14 and iSBC 86/30 board, respectively. The CPU, system clock, read/write memory, nonvolatile read only memory, I/O ports and drivers, serial communications interface, priority interrupt logic and programmable timers, all reside on the boards.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: CREDIT, Index, Intel, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

FUNCTIONAL DESCRIPTION

Central Processing Unit

The central processor for the iSBC 86/XX⁽¹⁾ boards is Intel's iAPX 86/10 (8086-2) CPU. A clock rate of 8 MHz is supported with a jumper selectable option of 5 MHz. The CPU architecture includes four 16-bit byte addressable data registers, two 16-bit memory base pointer registers and two 16-bit index registers, all accessed by a total of 24 operand addressing modes for comprehensive memory addressing and for support of the data structures required for today's structured, high level languages as well as assembly language.

⁽¹⁾ iSBC 86/XX designates both the iSBC 86/14 and iSBC 86/30 CPU boards.

Instruction Set

The 8086 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions.

For enhanced numerics processing capability, the iSBC 337 MULTIMODULE Numeric Data Processor extends the iAPX 86/10 architecture and data set. Over 60 numeric instructions offer arithmetic, trigonometric, transcendental, logarithmic and ex-

ponential instructions. Supported data types include 16, 32, and 64-bit integer, and 32 and 64-bit floating point, 18-digit packed BCD and 80-bit temporary.

Architectural Features

A 6-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 750 nsec minimum instruction cycle to 250 nsec for queued instructions. The stack-oriented architecture readily supports modular programming by facilitating fast, simple, inter-module communication, and other programming constructs needed for asynchronous real-time systems. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions.

RAM Capabilities

The iSBC 86/14 and iSBC 86/30 microcomputers contain 32K bytes and 128K bytes of dual-port dynamic RAM, respectively. In addition, on-board

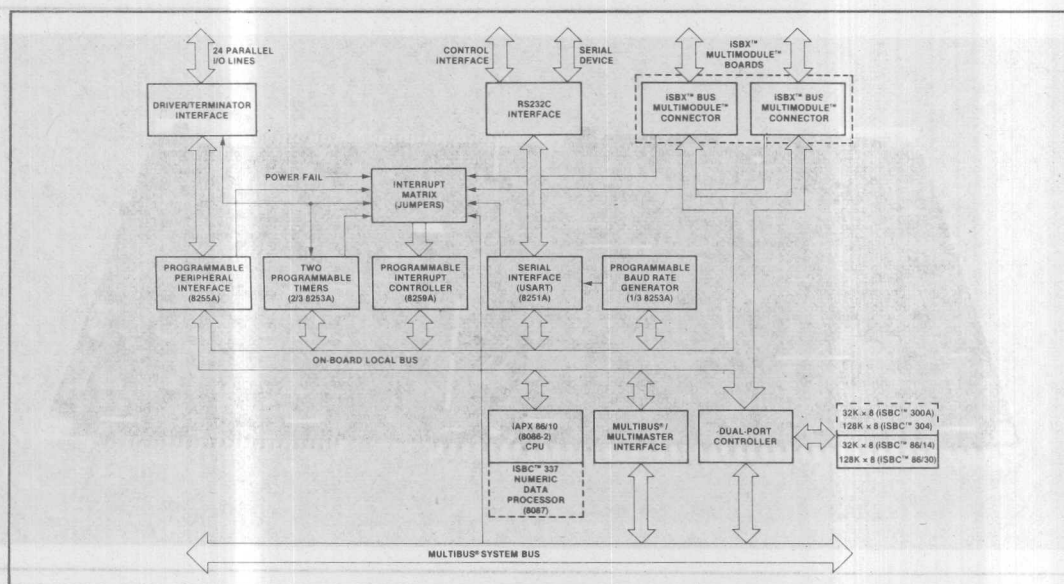


Figure 1. iSBC™ 86/XX Block Diagram

RAM may be doubled on each microcomputer by optionally adding RAM MULTIMODULE boards. The on-board RAM may be expanded to 256K bytes with the iSBC 304 MULTIMODULE Board mounted onto the iSBC 86/30 board. Likewise, the iSBC 86/14 microcomputer may be expanded to 64K bytes with the iSBC 300A MULTIMODULE option. The dual-port controller allows access to the on-board RAM (including RAM MULTIMODULE options) from the iSBC 86/XX boards and from any other MULTIBUS master via the system bus. Segments of on-board RAM may be configured as a private resource, protected from MULTIBUS system access. The amount of memory allocated as a private resource may be configured in increments of 25% of the total on-board memory ranging from 0% to 100% (optional RAM MULTIMODULE boards double the increment size). These features allow the multiprocessor systems to establish local memory for each processor and shared system memory configurations where the total system memory size (including local on-board memory) can exceed one megabyte without addressing conflicts.

EPROM Capabilities

Four 28-pin sockets are provided for the use of Intel 2716s, 2732As, 2764s, 27128s, and their respective ROMs. When using 27128s, the on-board EPROM capacity is 64K bytes. Other JEDEC standard pinout devices are also supported, including byte-wide static RAMs.

Parallel I/O Interface

The iSBC 86/XX Single Board Computers contain 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral Interface. The system software is used to configure the I/O lines in any combination of unidirectional

input/output and bidirectional ports indicated in Table 1. In order to take advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators, allowing the selection of the appropriate combination of optional line drivers and terminators with the required drive/termination characteristics. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector.

Serial I/O

A programmable communications interface using the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 86/XX boards. A software selectable baud rate generator provides the USART with all common communication frequencies. The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The RS232C command lines, serial data lines and signal ground line are brought out to a 26-pin edge connector.

Programmable Timers

The iSBC 86/XX boards provide three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be indepen-

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation				Control	
		Unidirectional			Bidirectional		
		Input		Output			
		Latched	Latched & Strobed	Latched			Latched & Strobed
1	8	X	X	X	X		
2	8	X	X	X	X		
3	4	X		X		X ¹	
	4	X		X		X ¹	

NOTE:

1. Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

dently routed to the 8259A Programmable Interrupt Controller and to the I/O terminators associated with the 8255A to allow external devices or an 8255A port to gate the timer or to count external events. The third interval timer in the 8253 provides the programmable baud rate generator for the iSBC 86/XX boards' RS232C USART serial port. The system software configures each timer independently to select the desired function. Seven functions are available as shown in Table 2. The contents of each counter may be read at any time during system operation.

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counter "window" has been enabled or an interrupt may be generated after N events occur in the system.

iSBX™ MULTIMODULE™ On-Board Expansion

Two 8/16-bit iSBX MULTIMODULE connectors are provided on the iSBC 86/XX microcomputers. Through these connectors, additional on-board I/O functions may be added. iSBX MULTIMODULE

boards optimally support functions provided by VLSI peripheral components such as additional parallel and serial I/O, analog I/O, small mass storage device controllers (e.g., cassettes and floppy disks), and other custom interfaces to meet specific needs. By mounting directly on the single board computer, less interface logic, less power, simpler packaging, higher performance, and lower cost result when compared to other alternatives such as MULTIBUS form factor compatible boards. The iSBX connectors on the iSBC 86/XX boards provide all signals necessary to interface to the local on-board bus, including 16 data lines for maximum data transfer rates. iSBX MULTIMODULE boards designed with 8-bit data paths and using the 8-bit iSBX connector are also supported on the iSBC 86/XX microcomputers. A broad range of iSBX MULTIMODULE options are available in this family from Intel. Custom iSBX modules may also be designed for use on the iSBC 86/XX boards. An iSBX bus interface specification and iSBX connectors are available from Intel.

MULTIBUS® SYSTEM BUS AND MULTIMASTER CAPABILITIES

Overview

The MULTIBUS system bus is Intel's industry standard microcomputer bus structure. Both 8 and 16-bit single board computers are supported on the MULTIBUS structure with 24 address and 16 data lines. In its simplest application, the MULTIBUS system bus allows expansion of functions already contained on a single board computer (e.g., memory and digital I/O). However, the MULTIBUS structure also allows very powerful distributed processing configurations with multiple processors and intelligent slave I/O, and peripheral boards capable of solving the most demanding microcomputer applications. The MULTIBUS system bus is supported with a broad array of board level products, LSI interface components, detailed published specifications and application notes.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. Memory may be expanded by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be added with digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers, or

hard disk controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

Multimaster Capabilities

For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication of the system bus), the iSBC 86/XX boards provide full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 86/XX boards or other bus masters, including iSBC 80 family MULTIBUS compatible 8-bit single board computers to share the system bus using a serial (daisy chain) priority scheme and allows up to 16 masters to share the MULTIBUS system bus with an external parallel priority decoder. In addition to the multiprocessing configurations made possible with multimaster capability, it also provides a very efficient mechanism for all forms of DMA (Direct Memory Access) transfers.

Interrupt Capability

The iSBC 86/XX boards provide 9 vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line which is directly tied to the 8086 CPU. This interrupt is typically used for signaling catastrophic events (e.g., power failure). The Intel 8259A Programmable Interrupt Controller (PIC) provides control and vectoring for the next eight interrupt levels. As shown in Table 3, a selection of four priority processing modes is available for use in designing request processing configurations to match system requirements for efficient interrupt servicing with minimal latencies. Operating mode and priority assignments may be reconfigured dynamically via software at any time

during system operation. The PIC accepts interrupt requests from all on-board I/O resources and from the MULTIBUS system bus. The PIC then resolves requests according to the selected mode and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked via software, by storing a single byte in the interrupt mask register of the PIC. In systems requiring additional interrupt levels, slave 8259A PICs may be interfaced via the MULTIBUS system bus, to generate additional vector addresses, yielding a total of 65 unique interrupt levels.

Interrupt Request Generation

Interrupt requests to be serviced by the iSBC 86/XX boards may originate from 28 sources. Table 4 includes a list of devices and functions supported by interrupts. All interrupt signals are brought to the interrupt jumper matrix where any combination of interrupt sources may be strapped to the desired interrupt request level on the 8259A PIC or the NMI input to the CPU directly.

Power-Fail Control and Auxiliary Power

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the iSBC 635 and iSBC 640 Power Supply or equivalent, to initiate an orderly shut down of the system in the event of a power failure. Additionally, an active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power-down sequences. An auxiliary power bus is also provided to allow separate power to RAM for systems requiring battery back-up of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

System Development Capabilities

The development cycle of iSBC 86/XX products can be significantly reduced and simplified by using either the System 86/330 or the Intellec Series Microcomputer Development Systems. The Assembler, Locating Linker, Library Manager, Text Editor and System Monitor are all supported by the ISIS-II disk-based operating system. To facilitate conversion of 8080A/8085A assembly language programs to run on the iSBC 86/XX boards, CONV-86 is available under the ISIS-II operating system.

Table 3. Programmable Interrupt Modes

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

IN-CIRCUIT EMULATOR

The Intel ICE-86 In-Circuit Emulator provides the necessary link between the software development environment provided by the Intel system and the "target" iSBC 86/XX execution system. In addition to providing the mechanism for loading executable code and data into the iSBC 86/XX boards, the ICE-86 In-Circuit Emulator provides a sophisticated command set to assist in debugging software and final integration of the user hardware and software.

PL/M-86

Intel's system's implementation language, PL/M-86, is standard in the System 86/330 and is also available as an Intel Microcomputer Development System option. PL/M-86 provides the capability to program in algorithmic language and eliminates the need to manage register usage or allocate memory while still allowing explicit control of the system's resources when needed. FORTRAN 86 and PASCAL 86 are also available on Intel or 86/330 systems.

Run-Time Support

Intel also offers two run-time support packages; iRMX 88 Realtime Multitasking Executive and the iRMX 86 Operating System. The iRMX 88 executive is a simple, highly configurable and efficient foundation for small, high performance applications. Its multitasking structure establishes a solid foundation for modular system design and provides task scheduling and management, intertask communication and synchronization, and interrupt servicing for a variety of peripheral devices. Other configurable options include terminal handlers, disk file system, debuggers and other utilities. The iRMX 86 Operating System is a high functional operating system with a very rich set of features and options based on an object-oriented architecture. In addition to being modular and configurable, functions beyond the nucleus include a sophisticated file management and I/O system, and powerful human interface. Both packages are easily customized and extended by the user to match unique requirements.

Table 4. Interrupt Request Sources

Device	Function	Number of Interrupts
MULTIBUS® interface	Requests from MULTIBUS® resident peripherals or other CPU boards	8; may be expanded to 64 with slave 8259A PICs on MULTIBUS® boards
8255A Programmable Peripheral Interface	Signals input buffer full or output buffer empty; also BUS INTR OUT general purpose interrupt from driver/terminator sockets	3
8251A USART	Transmit buffer empty and receive buffer full	2
8253 Timers	Timer 0, 1 outputs; function determined by timer mode	2
iSBX™ connectors	Function determined by iSBX™ MULTIMODULE™ board	4 (2 per iSBX™ connector)
Bus fail safe timer	Indicates addressed MULTIBUS® resident device has not responded to command within 6 msec	1
Power fail interrupt	Indicates AC power is not within tolerance	1
Power line clock	Source of 120 Hz signal from power supply	1
External interrupt	General purpose interrupt from auxiliary (P2) connector on backplane	1
iSBC™ 337 MULTIMODULE™ Numeric Data Processor	Indicates error or exception condition	1
Parity error	Indicates on-board RAM parity error from iSBC™ 303 parity MULTIMODULE™ board (iSBC™ 86/14 option)	1
Edge-level conversion	Converts edge triggered interrupt request to level interrupt	1
OR-gate matrix	Outputs of OR-gates on-board for multiple interrupts	2

SPECIFICATIONS

Word Size

INSTRUCTION — 8, 16, 24, or 32 bits

DATA — 8, 16 bits

System Clock

5.00 MHz or 8.00 MHz $\pm 0.1\%$ (jumper selectable)

Cycle Time

BASIC INSTRUCTION CYCLE

8 MHz — 750 ns

— 250 ns (assumes instruction in the queue)

5 MHz — 1.2 μ sec

— 400 ns (assumes instruction in the queue)

NOTE: Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles).

Memory Cycle Time

RAM — 750 ns

EPROM — Jumper selectable from 500 ns to 875 ns

Memory Capacity/Addressing

ON-BOARD EPROM

Device	Total Capacity	Address Range
2716	8K bytes	FE000–FFFFFH
2732A	16K bytes	FC000–FFFFFH
2764	32K bytes	F8000–FFFFFH
27128	64K bytes	F0000–FFFFFH

NOTE: iSBC 86/XX EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs.

ON-BOARD RAM

Board	Total Capacity	Address Range
iSBC 86/14	32K bytes	0–07FFFFH
iSBC 86/30	128K bytes	0–1FFFFFH

WITH MULTIMODULE™ RAM

Board	Total Capacity	Address Range
iSBC 300A (with iSBC 86/14)	64K bytes	0–0FFFFFH
iSBC 304 (with iSBC 86/30)	256K bytes	0–3FFFFFH

I/O Capacity

PARALLEL — 24 programmable lines using one 8255A.

SERIAL — 1 programmable line using one 8251A

ISBX™ MULTIMODULE™ — 2 iSBX boards

Serial Communications Characteristics

SYNCHRONOUS — 5–8 bit characters; internal or external character synchronization; automatic sync insertion

ASYNCHRONOUS — 5–8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection

BAUD RATES

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)	
	Synchronous	Asynchronous
153.6	—	+ 16 + 64
76.8	—	9600 2400
38.4	38400	4800 1200
19.2	19200	2400 600
9.6	9600	1200 300
4.8	4800	600 150
2.4	2400	300 75
1.76	1760	150 —
		110 —

NOTE: Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register (8253 Timer 2).

Timers

INPUT FREQUENCIES

Reference: 2.46 MHz $\pm 0.1\%$ (0.041 μ sec period, nominal); or 153.60 kHz $\pm 0.1\%$ (6.51 μ sec period, nominal)

NOTE: Above frequencies are user selectable.

Event Rate: 2.46 MHz max

OUTPUT FREQUENCIES/TIMING INTERVALS

Function	Single Timer/Counter		Dual Timer/Counter (Cascaded)	
	Min	Max	Min	Max
Real-time Interrupt	1.63 μ s	427.1 ms	3.26s	466.50 min
Programmable one-shot	1.63 μ s	427.1 ms	3.26s	466.50 min
Rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Square-wave rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Software triggered strobe	1.63 μ s	427.1 ms	3.26s	466.50 min
Hardware triggered strobe	1.63 μ s	427.1 ms	3.26s	466.50 min
Event counter	—	2.46 MHz	—	—

Interfaces

MULTIBUS® — All signals TTL compatible

ISBX™ BUS — All signals TTL compatible

PARALLEL I/O — All signals TTL compatible

SERIAL I/O — RS232C compatible, configurable as a data set or data terminal

TIMER — All signals TTL compatible

INTERRUPT REQUESTS — All TTL compatible

Connectors

Interface	Double-Sided Pins	Centers (in.)	Mating Connectors
MULTIBUS® System	86	0.156	Viking 3KH43/9AMK12 Wire Wrap
iSBX™ Bus 8-Bit Data	36	0.1	iSBX 960-5
Parallel I/O (2)	50	0.1	3M 3415-000 Flat or TI H312125 Pins
Serial I/O	26	0.1	3M 3462-0001 Flat or AMP 88106-1 Flat

Line Drivers and Terminators

I/O DRIVERS — The following line drivers are all compatible with the I/O driver sockets on the iSBC 86/05 board

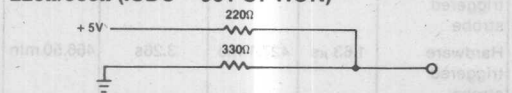
Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

NOTE: I = inverting; NI = non-inverting; OC = open collector.

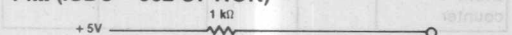
Port 1 of the 8255A has 20 mA totem-pole bidirectional drivers and 1 k Ω terminators

I/O TERMINATORS — 220 Ω /330 Ω divider or 1 k Ω pullup

220 Ω /330 Ω (iSBC™ 901 OPTION)



1 k Ω (iSBC™ 902 OPTION)



MULTIBUS® Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-State	32
Address	Tri-State	32
Commands	Tri-State	32
Bus Control	Open Collector	20

Physical Characteristics

WIDTH — 12.00 in. (30.48 cm)

HEIGHT — 6.75 in. (17.15 cm)

DEPTH — 0.70 in. (1.78 cm)

WEIGHT — 14 oz (388 gm)

Environmental Characteristics

OPERATING TEMPERATURE — 0°C to 55°C

RELATIVE HUMIDITY — to 90% (without condensation)

Electrical Characteristics

DC POWER REQUIREMENTS

Configuration	Current Requirements (All Voltages $\pm 5\%$)		
	+ 5V	+ 12V	- 12V
Without EPROM ¹	5.1A	25 mA	23 mA
RAM only ²	600 mA	—	—
With 8K EPROM ³ (using 2716)	5.4A	25 mA	23 mA
With 16K EPROM ³ (using 2732A)	5.5A	25 mA	23 mA
With 32K EPROM ³ (using 2764)	5.6A	25 mA	23 mA

NOTES:

- Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators.
- RAM chips powered via auxiliary power bus in power-down mode.
- Includes power required for 4 ROM/EPROM chips, and I/O terminators installed for 16 I/O lines; all terminator inputs low.

Environmental Characteristics

OPERATING TEMPERATURE — 0°C to 55°C

RELATIVE HUMIDITY — to 90% (without condensation)

Reference Manual

144044-001 — iSBC 86/14 and iSBC 86/30 Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

SBC 86/14 Single Board Computer

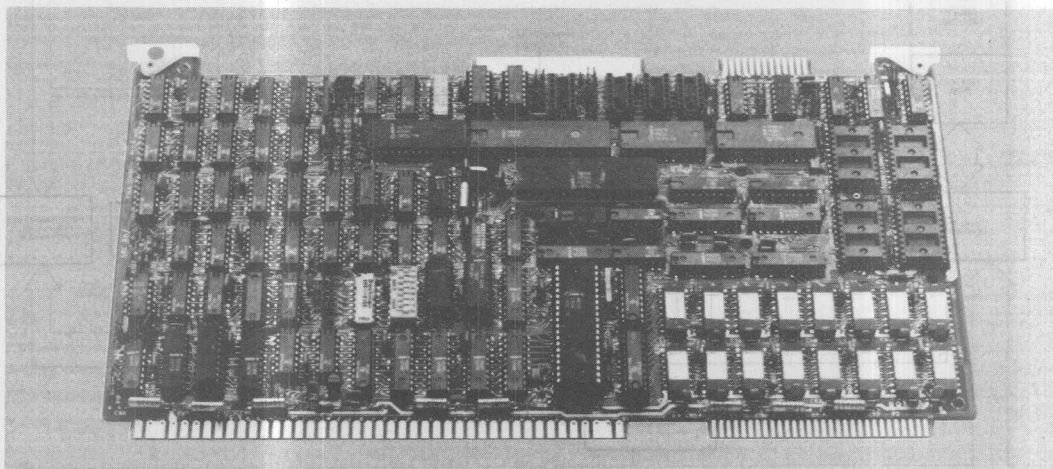
SBC 86/30 Single Board Computer



iSBC™ 86/12A or (pSBC 86/12A*) SINGLE BOARD COMPUTER

- 8086 16-bit HMOS microprocessor central processor unit
- 32K bytes of dual-port read/write memory expandable on-board to 64K bytes with on-board refresh
- Sockets for up to 16K bytes of read only memory expandable on-board to 32K bytes
- System memory expandable to 1 megabyte
- 24 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators
- Programmable synchronous/asynchronous RS232C compatible serial interface with software selectable baud rates
- Two programmable 16-bit BCD or binary timers/event counters
- 9 levels of vectored interrupt control, expandable to 65 levels
- Auxiliary power bus and power fail interrupt control logic for read/write memory battery backup
- MULTIBUS® interface for multimaster configurations and system expansion
- Compatible with iSBC 337 MULTIMODULE™ Numeric Data Processor
- Compatible with iSBC 80 family single board computers, memory, digital and analog I/O, and peripheral controller boards

The iSBC 86/12A Single Board Computer is a member of Intel's complete line of OEM microcomputer systems which take full advantage of Intel's LSI technology to provide economical self-contained computer based solutions for OEM applications. The iSBC 86/12A board is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, read/write memory, nonvolatile read only memory, I/O ports and drivers, serial communications interface, priority interrupt logic and programmable timers, all reside on the board. Full MULTIBUS interface logic is included to offer compatibility with the Intel OEM Microcomputer Systems family of Single Board Computers, expansion memory options, digital and analog I/O expansion boards and peripheral controllers.



*Same product, manufactured by Intel Puerto Rico, Inc.

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Index, Intel, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, iRMX, UPI, uScope, Promware, MCS, ICE, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1980, 1981

FUNCTIONAL DESCRIPTION

Central Processing Unit

The central processor for the iSBC 86/12A board is Intel's 8086, a powerful 16-bit HMOS device. The 225 sq. mil chip contains 29,000 transistors and has a clock rate of 5MHz. The architecture includes four (4) 16-bit byte addressable data registers, two (2) 16-bit memory base pointer registers and two (2) 16-bit index registers, all accessed by a total of 24 operand addressing modes for complex data handling and very flexible memory addressing.

Instruction Set — The 8086 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions. In addition, the iSBC 337 MULTIMODULE Numeric Data Processor may be installed to add over 60 numeric instructions and hardware support for multiple precision integer and floating point data types.

Architectural Features — A 6-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 1.2μsec minimum instruction cycle to 400 nsec for queued instructions. The stack oriented architecture facilitates nested subroutines and co-routines, reentrant

code and powerful interrupt handling. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K-bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions.

Bus Structure

The iSBC 86/12A microcomputer has three buses: an internal bus for communicating with on-board memory and I/O options, the MULTIBUS system bus for referencing additional memory and I/O options, and the dual-port bus which allows access to RAM from the on-board CPU and the MULTIBUS system bus. Local (on-board) accesses do not require MULTIBUS communication, making the system bus available for use by other MULTIBUS masters (i.e. DMA devices and other single board computers transferring to additional system memory). This feature allows true parallel processing in a multiprocessor environment. In addition, the MULTIBUS interface can be used for system expansion through the use of other 8- and 16-bit iSBC computers, memory and I/O expansion boards.

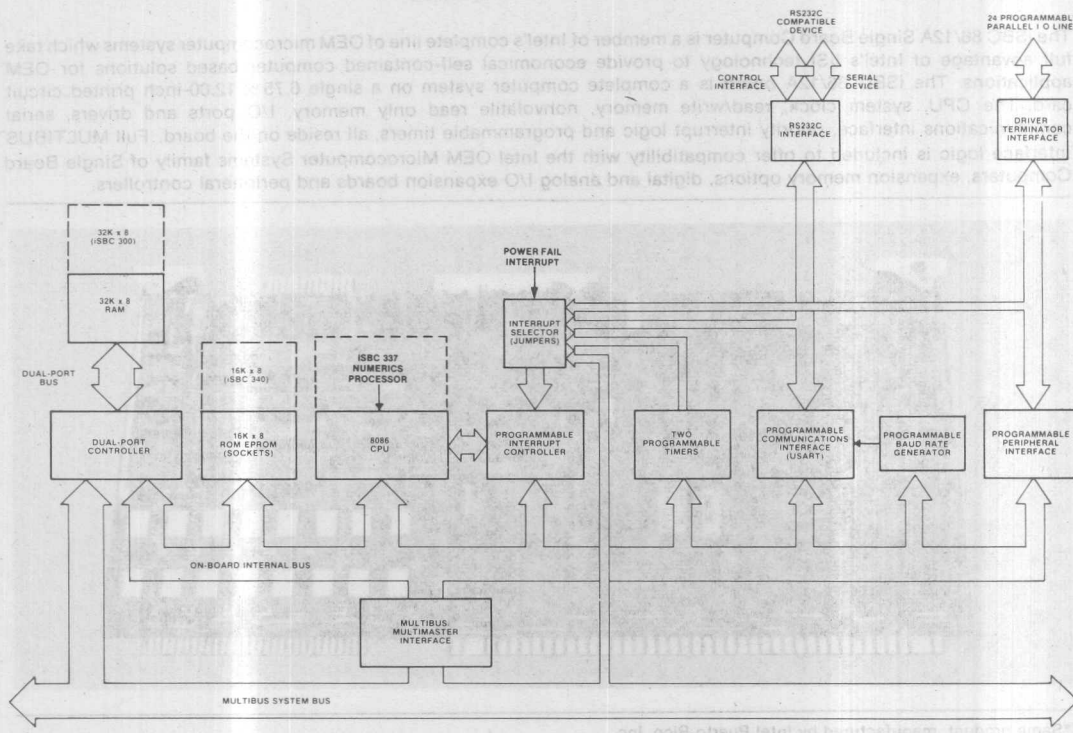


Figure 1. iSBC 86/12A Single Board Computer Block Diagram

RAM Capabilities

The iSBC 86/12A microcomputer contains 32K bytes of dynamic read/write memory using 16K-bit 2117 RAMs. In addition, the on-board RAM complement may be expanded to 64K bytes with the iSBC 300 32K-byte MULTIMODULE RAM option. Power for the on-board RAM and refresh circuitry may be optionally provided on an auxiliary power bus, and memory protect logic is included for RAM battery backup requirements. The iSBC 86/12A board contains a dual-port controller which allows access to the on-board RAM (32K bytes or 64K bytes when the iSBC 300 module is included with the iSBC 86/12A board) from the iSBC 86/12A CPU and from any other MULTIBUS master via the system bus. The dual-port controller allows 8- and 16-bit accesses from the MULTIBUS system bus, and the on-board CPU transfers data to RAM over a 16-bit data path. Priorities have been established such that memory refresh is guaranteed by the on-board refresh logic and that the on-board CPU has priority over MULTIBUS system bus requests for access to RAM. The dual-port controller includes independent addressing logic for RAM access from the on-board CPU and from the MULTIBUS system bus. The on-board CPU will always access RAM starting at location 00000H. Address jumpers allow on-board RAM to be located starting on any 8K-byte boundary within a 1 megabyte address range for accesses from the MULTIBUS system bus. In conjunction with this feature, the iSBC 86/12A microcomputer has the ability to protect on-board memory from MULTIBUS access to any contiguous 8K-byte segments (or 16K-byte segments with iSBC 300 module). These features allow the multi-processor systems to establish local memory for each processor and shared system (MULTIBUS) memory configurations where the total system memory size (including local on-board memory) can exceed 1 megabyte without addressing conflicts.

EPROM Capabilities

Four sockets are provided for up to 16K bytes of non-volatile read only memory on the iSBC 86/12A board. EPROM may be added in 2K-byte increments up to a maximum of 4K bytes by using Intel® 2758 electrically

programmable ROMs (EPROMs); in 4K-byte increments up to 8K bytes by using Intel 2716 EPROMs; or in 8K-byte increments up to 16K bytes using Intel 2732 EPROMs. On-board EPROM is accessed via 16-bit data paths. On-board EPROM capacity may be expanded to 32K bytes with the addition of the iSBC 340 16K-byte MULTIMODULE EPROM option. It provides an additional four sockets for Intel 2732 EPROMs. With user modification of the iSBC 86/12A's on-board memory and MULTIBUS address decode, Intel 2758 and 2716 EPROMs may be optionally supported. System memory size is easily expanded by the addition of MULTIBUS system bus compatible memory boards available in the iSBC product family.

Parallel I/O Interface

The iSBC 86/12A single board computer contains 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral Interface. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports indicated in Table 1. Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector that mates with flat, woven, or round cable.

Serial I/O

A programmable communications interface using the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 86/12A board. A software selectable baud rate generator provides the USART with all common communication

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation				Control	
		Unidirectional					Bidirectional
		Input		Output			
		Latched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	4	X		X			X ¹
	4	X		X			X ¹

Note

1. Port of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

synchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The RS232C compatible interface on each board, in conjunction with the USART, provides a direct interface to RS232C compatible terminals, cassettes, and asynchronous and synchronous modems. The RS232C command lines, serial data lines, and signal ground line are brought out to a 26 pin edge connector that mates with RS232C compatible flat or round cable. The iSBC 530 Teletypewriter Adapter provides an optically isolated interface for those systems requiring a 20 mA current loop. The iSBC 530 unit may be used to interface the iSBC 86/12A board to teletypewriters or other 20 mA current loop equipment.

Programmable Timers

The iSBC 86/12A board provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller and to the I/O line drivers associated with the 8255A Programmable Peripheral Interface, or may be routed as inputs to the 8255A chip. The gate/trigger inputs may be routed to I/O terminators associated with the 8255A or as output connections from the 8255A. The third interval timer in the 8253 provides the programmable baud rate generator for the iSBC 86/12A board RS232C USART serial port. In utilizing the iSBC 86/12A board the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given time delay or count is needed, software commands to the programmable timers/event counters select the desired function. Seven functions are available, as shown in Table 2. The contents of each counter may be read at any time during system operation with simple read operations for event counting applications, and special commands are included so that the contents can be read "on the fly".

MULTIBUS System Bus and Multimaster Capabilities

The MULTIBUS system bus features asynchronous data transfers for the accommodation of devices with various transfer rates while maintaining maximum throughput. Twenty address lines and sixteen separate data lines eliminate the need for address/data multiplexing/demultiplexing logic used in other systems, and allow for data transfer rates up to 5 megawords/sec. A failsafe timer is included in the iSBC 86/12A board which can be used to generate an interrupt if an addressed device does not respond within 6 msec.

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counting "window" has been enabled or an interrupt may be generated after N events occur in the system.

Multimaster Capabilities — The iSBC 86/12A board is a full computer on a single board with resources capable of supporting a great variety of OEM system requirements. For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication over the system bus), the iSBC 86/12A board provides full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 86/12A boards or other bus masters, including iSBC 80 family MULTIBUS compatible 8-bit single board computers, to share the system bus in serial (daisy chain) priority fashion and up to 16 masters to share the MULTIBUS system bus with the addition of an external priority network. The MULTIBUS arbitration logic operates synchronously with a MULTIBUS clock (provided by the iSBC 86/12A board or optionally provided directly from the MULTIBUS) while data is transferred via a handshake between the master and slave modules. This allows different speed controllers to share resources on the same bus, and transfers via the bus proceed asynchronously. Thus, transfer speed is dependent on transmitting and

receiving devices only. This design prevents slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed peripheral control, but are by no means limited to these three.

Interrupt Capability

The iSBC 86/12A board provides 9 vectored interrupt levels. The highest level is the NMI (Non-maskable Interrupt) line which is directly tied to the 8086 CPU. This interrupt cannot be inhibited by software and is typically used for signalling catastrophic events (i.e., power failure). On servicing this interrupt, program control will be implicitly transferred through location 00008H. The Intel 8259A Programmable Interrupt Controller (PIC) provides vectoring for the next eight interrupt levels. As shown in Table 3, a selection of four priority processing modes is available to the systems designer for use in designing request processing configurations to match system requirements. Operating mode and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from the programmable parallel and serial I/O interfaces, the programmable timers, the system bus, or directly from peripheral equipment. The PIC then determines which of the incoming requests is of the highest priority, determines whether this request is of higher priority than the level currently being serviced, and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. The PIC generates a unique memory address for each interrupt level. These addresses are equally spaced at 4 byte intervals. This 32-byte block may begin at any 32-byte boundary in the lowest 1K-bytes of memory,* and contains unique instruction pointers and code segment offset values (for expanded memory operation) for each interrupt level. After acknowledging an interrupt and obtaining a device identifier byte from the 8259A PIC, the CPU will store its status flags on the stack and execute an indirect CALL instruction through the vector location (derived from the device identifier) to the interrupt service routine. In systems requiring additional interrupt levels, slave 8259A PIC's may be interfaced via the MULTIBUS system bus, to generate additional vector addresses, yielding a total of 65 unique interrupt levels.

Interrupt Request Generation — Interrupt requests may originate from 18 sources. Two jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface when a byte of

***Note:** The first 32 vector locations are reserved by Intel for dedicated vectors. Users who wish to maintain compatibility with present and future Intel products should not use these locations for user-defined vector addresses.

Table 3. Programmable Interrupt Modes

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

information is ready to be transferred to the CPU (i.e., input buffer is full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). Two jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU (i.e., receive channel buffer is full, or a character is ready to be transmitted (i.e., transmit channel data buffer is empty). A jumper selectable request can be generated by each of the programmable timers. An additional interrupt request line may be jumpered directly from the parallel I/O driver terminator section. Eight prioritized interrupt request lines allow the iSBC 86/12A board to recognize and service interrupts originating from peripheral boards interfaced via the MULTIBUS system bus. The MULTIBUS fail safe timer of the iSBC 337 processor and the exception and error output signal also can be selected as interrupt sources.

Power-Fail Control

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the iSBC 635 and iSBC 640 Power Supply or equivalent.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. Memory may be expanded by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers, or hard disk controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

Note: Certain system restrictions may be incurred by the inclusion of some of the iSBC 80 family options in an iSBC 86/12A system. Consult the Intel OEM Microcomputer System Configuration Guide for specific data.

System Development Capabilities

The development cycle of iSBC 86/12A products can be significantly reduced by using the Intellec® series microcomputer development system. The Assembler, High Level Languages, Locating Linker, Library Manager, Text Editor and System Monitor are all supported by the ISIS-II disk-based operating system.

In-Circuit Emulator — ICE™-86 in-circuit emulator provides the necessary link between the software development environment provided by the Intellec system and the "target" iSBC 86/12A execution system. In addition to providing the mechanism for loading executable code and data into the iSBC 86/12A board, ICE-86 in-circuit emulator provides a sophisticated command set to assist in debugging software and final integration of the user hardware and software. ICE-86 in-circuit emulator maximizes the use of available development resources by

allowing Intellec resident resources (e.g., memory and peripherals) to be accessed by software running on the target iSBC 86/12A system. In addition, software can be executed without an iSBC 86/12A execution vehicle, in 2K bytes of RAM resident in the ICE-86 system itself. Symbolic references to instruction and data locations can be made through ICE-86 in-circuit emulator to allow the user to reference memory locations with assigned names.

PL/M-86 — Intel's high level programming language, PL/M-86, is also available as an Intellec Microcomputer Development System option. PL/M-86 provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M-86 programs can be written in a much shorter time than assembly language programs for a given application. PL/M-86 includes byte and word, integer, pointer and floating point (32-bit) data types and also includes conditional compilation and macro features.

SPECIFICATIONS

Word Size

Instruction — 8, 16, 24, or 32 bits

Data — 8, 16 bits

Cycle Time

Basic Instruction Cycle — 1.2 μ sec

— 400 nsec (assumes instruction in the queue)

Note:

Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles)

Memory Capacity

On-Board Read Only Memory — 16K bytes (sockets only); expandable to 32K bytes with iSBC 340 16K-byte MULTIMODULE EPROM option.

On-Board RAM — 32K bytes; expandable to 64K bytes with iSBC 300 32K-byte MULTIMODULE RAM option.

Off-Board Expansion — Up to 1 megabyte in user specified combinations of RAM and EPROM.

Note:

Read only memory may be added in 2K, 4K, or 8K-byte increments.

Memory Addressing

On-Board EPROM — FF000-FFFFH (using 2758 EPROMs); FE000-FFFFH (using 2716 EPROMs); FC000-FFFFH (using 2732 EPROMs); F8000-FFFFH (with iSBC 340 EPROM option and four additional 2732 EPROMs).

On-Board RAM — 32K bytes of dual port RAM. Optionally expandable to 64K bytes with iSBC 300 RAM option.

CPU Access — 32K bytes: 00000-07FFFH; 64K bytes: 00000-0FFFFH.

MULTIBUS Access — Jumper selectable for any 8K-byte boundary, but not crossing a 128K-byte boundary. Access for 8K, 16K, 24K or 32K (16K, 32K, 48K, 64K with iSBC 300 option) bytes may be selected for on-board CPU use only.

I/O Capacity

Parallel — 24 programmable lines using one 8255A.

Serial — 1 programmable line using one 8251A.

I/O Addressing

On-Board Programmable I/O

Port	8255A				USART	
	1	2	3	Control	Data	Control
Address	C8	CA	CC	CE	D8 or DC	DA or DE

Serial Communications Characteristics

Synchronous — 5—8 bit characters; internal or external character synchronization; automatic sync insertion.

Asynchronous — 5—8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection.

Baud Rates

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)	
	Synchronous	Asynchronous
153.6	—	+ 16 + 64
76.8	—	9600 2400
38.4	38400	4800 1200
19.2	19200	2400 600
9.6	9600	1200 300
4.8	4800	600 150
2.4	2400	300 75
1.76	1760	150 150
		110 —

Note:

Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register (8253 Timer 2).

Interrupts

Addresses for 8259A Registers (Hex notation I/O address space)

C0 or C4 Write: Initialization Command Word 1 (ICW1) and Operation Control Words 2 and 3 (OCW2 and OCW3)

Read: Status and Poll Registers

C2 or C6 Write: ICW2, ICW3, ICW4, OCW1 (Mask Register)

Read: OCW1 (Mask Register)

Note:

Several registers have the same physical address; sequence of access and one data bit of control word determine which register will respond.

Interrupt Levels — 8086 CPU includes a non-maskable interrupt (NMI) and a maskable interrupt (INTR). NMI interrupt is provided for catastrophic events such as power failure. NMI vector address is 00008. INTR interrupt is driven by on-board 8259A PIC, which provides 8-bit identifier of interrupting device to CPU. CPU multiplies identifier by four to derive vector address. Jumpers select interrupts from 18 sources without necessity of external hardware. PIC may be programmed to accommodate edge-sensitive or level-sensitive inputs.

Timers

Register Addresses (Hex notation, I/O address space)

D0 Timer 0

D2 Timer 1

D4 Timer 2

D6 Counter register

Note:

Timer counts are loaded as two sequential output operations to same address as given.

Input Frequencies

Reference: 2.46 MHz \pm 0.1% (0.041 μ s period, nominal); 1.23 MHz \pm 0.1% (0.81 μ s period, nominal); or 153.60 kHz \pm 0.1% (6.51 μ s period nominal).

Note:

Above frequencies are user selectable.

Event Rate: 2.46 MHz max

Output Frequencies/Timing Intervals

Function	Single Timer/Counter		Dual Timer/Counter (Two Timers Cascaded)	
	Min	Max	Min	Max
Real-time interrupt	1.63 μ s	427.1 ms	3.26 s	466.50 min
Programmable one-shot	1.63 μ s	427.1 ms	3.26 s	466.50 min
Rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Square-wave rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Software triggered strobe	1.63 μ s	427.1 ms	3.26 s	466.50 min
Hardware triggered strobe	1.63 μ s	427.1 ms	3.26 s	466.50 min
Event counter	—	2.46 MHz	—	—

Interfaces

MULTIBUS — All signals TTL compatible

Parallel I/O — All signals TTL compatible

Interrupt Requests — All TTL compatible

Timer — All signals TTL compatible

Serial I/O — RS232C compatible, data set configuration

System Clock (8086 CPU)

5.00 MHz \pm 0.1%

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM for systems requiring battery backup of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

Connectors

Interface	Pins (qty)	Centers (in.)	Mating Connectors
Bus	86	0.156	VIKING 3KH43/9AMK12
Parallel I/O	50	0.1	3M 3415-000
Serial I/O	26	0.1	3M 3462-000

Memory Protect

An active low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power down sequences.

Line Drivers and Terminators

I/O Drivers — The following line drivers are all compatible with the I/O driver sockets on the iSBC 86/12A board

Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

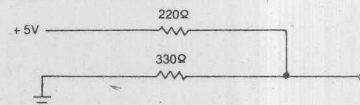
Note:

I = inverting; NI = non-inverting; OC = open collector.

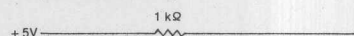
Port 1 of the 8255A has 20 mA totem-pole bidirectional drivers and 1 k Ω terminators.

I/O Terminators — 220 Ω /330 Ω divider or 1 k Ω pullup

220 Ω /330 Ω (iSBC 901 OPTION)



1 K Ω (iSBC 902 OPTION)



Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-state	50
Address	Tri-state	50
Commands	Tri-state	32

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.70 in. (1.78 cm)

Weight — 19 oz. (539 gm)

Electrical Characteristics
DC Power Requirements

Configuration	Current Requirements			
	V _{CC} = +5V ± 5% (max)	V _{DD} = +12V ± 5% (max)	V _{BB} = -5V ± 5% (max)	V _{AA} = -12V ± 5% (max)
Without EPROM ¹	5.2A	350 mA	—	40 mA
RAM Only ³	390 mA	40 mA	1.0 mA	—
With iSBC 530 ⁴	5.2A	450 mA	—	140 mA
With 4K EPROM ⁵ (using 2758)	5.5A	350 mA	—	40 mA
With 8K EPROM ⁵ (using 2716)	5.5A	350 mA	—	40 mA
With 16K EPROM ⁵ (using 2732)	5.4A	350 mA	—	40 mA

Notes:

- Does not include power for optional EPROM, I/O drivers, and I/O terminators.
- Does not include power required for optional EPROM, I/O drivers, and I/O terminators.
- RAM chips powered via auxiliary power bus.
- Does not include power for optional EPROM, I/O drivers, and I/O terminators. Power for iSBC 530 is supplied via serial port connector.
- Includes power required for four EPROM chips, and I/O terminators installed for 16 I/O lines; all terminator inputs low.

ORDERING INFORMATION

Part Number	Description
SBC 86/12A	Single Board Computer with 32K bytes RAM

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Relative Humidity — to 90% (without condensation)

Reference Manual

9803074-01 — iSBC 86/12A Single Board Computer Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

iSBC 86/05™ SINGLE BOARD COMPUTER

- iAPX 86/10 (8086-2) Microprocessor with 5 or 8 MHz CPU clock
- Fully software compatible with iSBC 86/12A Single Board Computer
- Optional iAPX 86/20 Numeric Data Processor with iSBC 337 MULTIMODULE Processor
- 8K bytes of static RAM; expandable on-board to 16K bytes
- Sockets for up to 64K bytes of JEDEC 24/28-pin standard memory devices; expandable on-board to 128K bytes
- Two iSBX™ bus connectors
- 24 programmable parallel I/O lines
- Programmable synchronous/asynchronous RS232C compatible serial interface with software selectable baud rates
- Two programmable 16-bit BCD or binary timers/event counters
- 9 Levels of vectored interrupt control, expandable to 65 levels
- MULTIBUS interface for multimaster configurations and system expansion
- Supported by a complete family of single board computers, memory, digital and analog I/O, peripheral controllers, packaging and software

The iSBC 86/05 Single Board Computer is a member of Intel's complete line of OEM microcomputer systems which take full advantage of Intel's technology to provide economical, self-contained, computer-based solutions for OEM applications. The iSBC 86/05 board is a complete computer system on a single 6.75 × 12.00-in. printed circuit card. The CPU, system clock, read/write memory, nonvolatile read only memory, I/O ports and drivers, serial communications interface, priority interrupt logic and programmable timers, all reside on the board. The large control storage capacity makes the iSBC 86/05 board ideally suited for control-oriented applications such as process control, instrumentation, industrial automation, and many others.

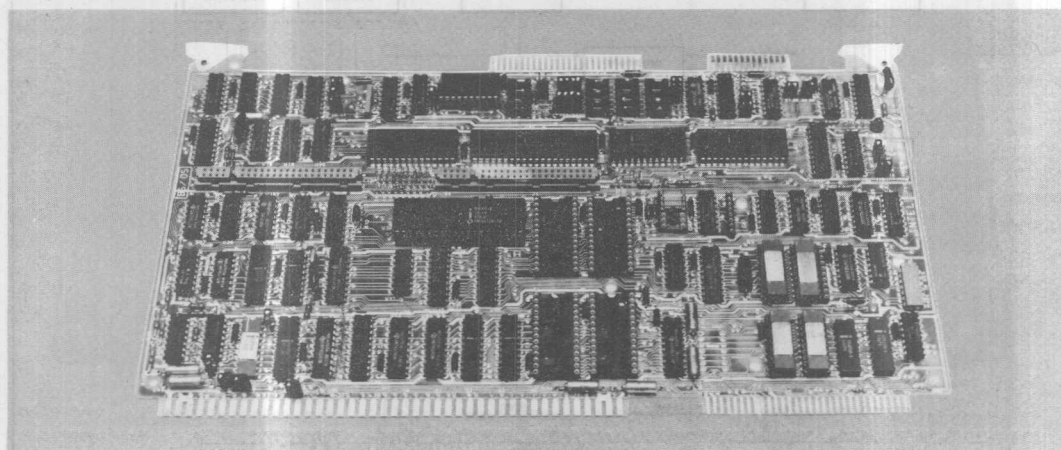


Figure 1. iSBC 86/05 Block Diagram

FUNCTIONAL DESCRIPTION

Central Processing Unit

The central processor for the iSBC 86/05 board is Intel's iAPX 86/10 (8086-2) CPU. A clock rate of 8 MHz is supported with a jumper selectable option of 5 MHz. The CPU architecture includes four 16-bit byte addressable data registers, two 16-bit memory base pointer registers and two 16-bit index registers, all accessed by a total of 24 operand addressing modes for comprehensive memory addressing and for support of the data structures required for today's structured, high level languages as well as assembly language.

Instruction Set

The 8086 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions.

For enhanced numerics processing capability, the iSBC 337 MULTIMODULE Numeric Data Processor extends the iAPX 86/10 architecture and data set. Over 60 numeric instructions offer arithmetic, trigonometric, transcendental, logarithmic and ex-

ponential instructions. Supported data types include 16, 32, and 64-bit integer, and 32 and 64-bit floating point, 18-digit packed BCD and 80-bit temporary.

Architectural Features

A 6-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 750 nsec minimum instruction cycle to 250 nsec for queued instructions. The stack-oriented architecture readily supports modular programming by facilitating fast, simple, inter-module communication, and other programming constructs needed for asynchronous real-time systems. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions. All Intel languages support the extended memory capability, relieving the programmer of managing the megabyte memory space, yet allowing explicit control when necessary.

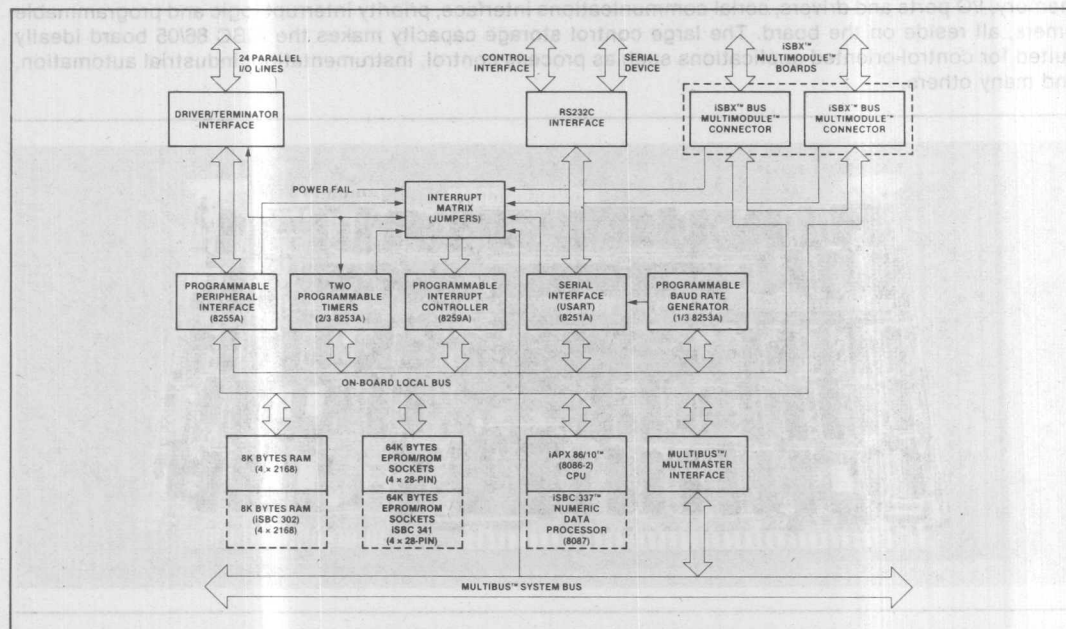


Figure 1. iSBC 86/05 Block Diagram

Memory Configuration

The iSBC 86/05 microcomputer contains 8K bytes of high-speed static RAM on-board. In addition, the on-board RAM may be expanded to 16K bytes with the iSBC 302 MULTIMODULE RAM option which mounts on the iSBC 86/05 board. All on-board RAM is accessed by the 8086-2 CPU with no wait states, yielding a memory cycle time of 500 nsec.

In addition to the on-board RAM, the iSBC 86/05 board has four 28-pin sockets, configured to accept JEDEC 24/28-pin standard memory devices. Up to 64K bytes of EPROM are supported in 16K-byte increments with Intel 27128 EPROMs. The iSBC 86/05 board is also compatible with the 2716, 2732A, and 2764 EPROMs offering expansion to 8, 16 and 32K bytes, respectively.

With the addition of the iSBC 341 MULTIMODULE EPROM option, the on-board capacity for these devices is doubled, providing up to 128K bytes of EPROM capacity on-board.

Parallel I/O Interface

The iSBC 86/05 Single Board Computer contains 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral Interface. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports indicated in Table 1. In order to take advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line

drivers and terminators, allowing the selection of the appropriate combination of optional line drivers and terminators with the required drive/termination characteristics. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector.

Serial I/O

A programmable communications interface using the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 86/05 board. A software selectable baud rate generator provides the USART with all common communication frequencies. The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The RS232C compatible interface on each board, in conjunction with the USART, provides a direct interface to RS232C compatible terminals, cassettes, and asynchronous and synchronous modems. The RS232C command lines, serial data lines and signal ground line are brought out to a 26-pin edge connector.

Programmable Timers

The iSBC 86/05 board provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation					Control
		Unidirectional				Bidirectional	
		Input		Output			
		Latched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	4	X		X			X ¹
	4	X		X			X ¹

NOTE:

1. Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller and to the I/O terminators associated with the 8255A to allow external devices or an 8255A port to gate the timer or to count external events. The third interval timer in the 8253 provides the programmable baud rate generator for the iSBC 86/05 board RS232C USART serial port. The system software configures each timer independently to select the desired function. Seven functions are available as shown in Table 2. The contents of each counter may be read at any time during system operation.

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counter "window" has been enabled or an interrupt may be generated after N events occur in the system.

iSBX MULTIMODULE On-Board Expansion

Two 8/16-bit iSBX MULTIMODULE connectors are provided on the iSBC 86/05 microcomputer. Through these connectors, additional on-board I/O functions may be added. iSBX MULTIMODULES optimally support functions provided by VLSI peripheral components such as additional parallel and serial I/O, analog I/O, small mass storage device controllers (e.g., cassettes and floppy disks), and other custom interfaces to meet specific needs. By mounting directly on the single board computer, less interface logic, less power, simpler packaging, higher performance, and lower cost result when compared to other alternatives such as MULTIBUS form factor compatible boards. The iSBX connectors on the iSBC 86/05 provide all signals necessary to interface to the local on-board bus, including 16 data lines for maximum data transfer rates. iSBX MULTIMODULE boards designed with 8-bit data paths and using the 8-bit iSBX connector are also supported on the iSBC 86/05 microcomputer. A broad range of iSBX MULTIMODULE options are available in this family from Intel. Custom iSBX modules may also be designed for use on the iSBC 86/05 board. An iSBX bus interface specification and iSBX connectors are available from Intel.

MULTIBUS™ SYSTEM BUS AND MULTIMASTER CAPABILITIES

Overview

The MULTIBUS system bus is Intel's industry standard microcomputer bus structure. Both 8 and 16-bit single board computers are supported on the MULTIBUS structure with 24 address and 16 data lines. In its simplest application, the MULTIBUS system bus allows expansion of functions already contained on a single board computer (e.g., memory and digital I/O). However, the MULTIBUS structure also allows very powerful distributed processing configurations with multiple processors and intelligent slave I/O, and peripheral boards capable of solving the most demanding microcomputer applications. The MULTIBUS system bus is supported with a broad array of board level products, LSI interface components, detailed published specifications and application notes.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS

compatible expansion boards. Memory may be expanded by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be added with digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers, or hard disk controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

Multimaster Capabilities

For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication of the system bus), the iSBC 86/05 board provides full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 86/05 boards or other bus masters, including iSBC 80 family MULTIBUS compatible 8-bit single board computers to share the system bus using a serial (daisy chain) priority scheme and allows up to 16 masters to share the MULTIBUS system bus with an external parallel priority decoder. In addition to the multiprocessing configurations made possible with multimaster capability, it also provides a very efficient mechanism for all forms of DMA (Direct Memory Access) transfers.

Interrupt Capability

The iSBC 86/05 board provides 9 vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line which is directly tied to the 8086 CPU. This interrupt is typically used for signaling catastrophic events (e.g., power failure). The Intel 8259A Programmable Interrupt Controller (PIC) provides control and vectoring for the next eight interrupt levels. As shown in Table 3, a selection of four priority processing modes is available for use in designing request processing configurations to match system requirements for efficient interrupt servicing with minimal latencies. Operating mode and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from all on-board I/O resources and from the MULTIBUS system bus. The PIC then resolves requests according to the selected mode and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked via software, by storing a single byte in the interrupt mask register of the PIC. In systems requiring additional interrupt levels, slave 8259A PICs may be interfaced via the

MULTIBUS system bus, to generate additional vector addresses, yielding a total of 65 unique interrupt levels.

Table 3. Programmable Interrupt Modes

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

Interrupt Request Generation

Interrupt requests to be serviced by the iSBC 86/05 board may originate from 24 sources. Table 4 includes a list of devices and functions supported by interrupts. All interrupt signals are brought to the interrupt jumper matrix where any combination of interrupt sources may be strapped to the desired interrupt request level on the 8259A PIC or the NMI input to the CPU directly.

Power-Fail Control and Auxiliary Power

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the iSBC 635 and iSBC 640 Power Supply or equivalent, to initiate an orderly shut down of the system in the event of a power failure. Additionally, an active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power-down sequences. An auxiliary power bus is also provided to allow separate power to RAM for systems requiring battery back-up of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

System Development Capabilities

The development cycle of iSBC 86/05 products can be significantly reduced and simplified by

using the Intellec Series Microcomputer Development Systems. The Assembler, Locating Linker, Library Manager, Text Editor and System Monitor are all supported by the ISIS-II disk-based operating system. To facilitate conversion of 8080A/8085A assembly language programs to run on the iSBC 86/05 board, CONV-86 is available under the ISIS-II operating system.

IN-CIRCUIT EMULATOR

The ICE-86 In-Circuit Emulator provides the necessary link between the software development environment provided by the Intellec system and the "target" iSBC 86/05 execution system. In addition to providing the mechanism for loading executable code and data into the iSBC 86/05 board, the ICE-86 In-Circuit Emulator provides a sophisticated command set to assist in debugging software and final integration of the user hardware and software.

PL/M-86

Intel's system's implementation language, PL/M-86, is also available as an Intellec Microcomputer Development System option. PL/M-86 provides the capability to program in algorithmic

language and eliminates the need to manage register usage or allocate memory while still allowing explicit control of the system's resources when needed.

Run-Time Support

Intel also offers two run-time support packages; iRMX 88 Realtime Multitasking Executive and the iRMX 86 Operating System. iRMX 88 is a simple, highly configurable and efficient foundation for small, high performance applications. Its multi-tasking structure establishes a solid foundation for modular system design and provides task scheduling and management, intertask communication and synchronization, and interrupt servicing for a variety of peripheral devices. Other configurable options include terminal handlers, disk file system, debuggers and other utilities. iRMX 86 is a high functional operating system with a very rich set of features and options based on an object-oriented architecture. In addition to being modular and configurable, functions beyond the nucleus include a sophisticated file management and I/O system, and powerful human interface. Both packages are easily customized and extended by the user to match unique requirements.

Table 4. Interrupt Request Sources

Device	Function	Number of Interrupts
MULTIBUS interface	Requests from MULTIBUS resident peripherals or other CPU boards	8; may be expanded to 64 with slave 8259A PICs on MULTIBUS boards
8255A Programmable Peripheral Interface	Signals input buffer full or output buffer empty; also BUS INTR OUT general purpose interrupt from driver/terminator sockets	3
8251A USART	Transmit buffer empty and receive buffer full	2
8253 Timers	Timer 0, 1 outputs; function determined by timer mode	2
iSBX connectors	Function determined by iSBX MULTIMODULE board	4 (2 per iSBX connector)
Bus fail safe timer	Indicates addressed MULTIBUS resident device has not responded to command within 6 msec	1
Power fail interrupt	Indicates AC power is not within tolerance	1
Power line clock	Source of 120 Hz signal from power supply	1
External interrupt	General purpose interrupt from auxiliary (P2) connector on backplane	1
iSBC 337 MULTIMODULE Numeric Data Processor	Indicates error or exception condition	1

SPECIFICATIONS

Word Size

INSTRUCTION — 8, 16, 24, or 32 bits

DATA — 8, 16 bits

System Clock

5.00 MHz or 8.00 MHz $\pm 0.1\%$ (Jumper selectable)

Cycle Time

BASIC INSTRUCTION CYCLE

At 8 MHz — 750 nsec

— 250 nsec (assumes instruction in the queue)

At 5 MHz — 1.2 μ sec

— 400 nsec (assumes instruction in the queue)

NOTES:

Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles).

Memory Cycle Time

RAM — 500 nsec (no wait states)

EPROM — Jumper selectable from 500 nsec to 875 nsec

Memory Capacity/Addressing

ON-BOARD EPROM

Device	Total Capacity	Address Range
2716	8K bytes	FE000-FFFF _H
2732A	16K bytes	FC000-FFFF _H
2764	32K bytes	F8000-FFFF _H
27128	64K bytes	F0000-FFFF _H

WITH iSBC 341 MULTIMODULE EPROM

Device	Total Capacity	Address Range
2716	16K bytes	FC000-FFFF _H
2732A	32K bytes	F8000-FFFF _H
2764	64K bytes	F0000-FFFF _H
27128	128K bytes	E0000-FFFF _H

NOTES:

iSBC 86/05 EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs; iSBC 341 sockets also support E²PROMs.

ON-BOARD RAM

8K bytes — 0-1FFF_H

WITH iSBC 302 MULTIMODULE RAM

16K bytes — 0-3FFF_H

I/O Capacity

PARALLEL — 24 programmable lines using one 8255A.

SERIAL — 1 programmable line using one 8251A

iSBX MULTIMODULE — 2 iSBX MULTIMODULE boards

Serial Communications Characteristics

SYNCHRONOUS — 5-8 bit characters; internal or external character synchronization; automatic sync insertion

ASYNCHRONOUS — 5-8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection

BAUD RATES

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)	
	Synchronous	Asynchronous
153.6	—	+ 16 + 64
76.8	—	9600 2400
38.4	38400	4800 1200
19.2	19200	2400 600
9.6	9600	1200 300
4.8	4800	600 150
2.4	2400	300 75
1.76	1760	150 —
		110 —

NOTES:

Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register (8253 Timer 2).

Timers

INPUT FREQUENCIES

Reference: 2.46 MHz $\pm 0.1\%$ (0.041 μ sec period, nominal); or 153.60 kHz $\pm 0.1\%$ (6.51 μ sec period, nominal)

NOTES:

Above frequencies are user selectable.

Event Rate: 2.46 MHz max

OUTPUT FREQUENCIES/TIMING INTERVALS

Function	Single Timer/Counter		Dual Timer/Counter (Two Timers Cascaded)	
	Min	Max	Min	Max
Real-time Interrupt	1.63 μ s	427.1 ms	3.26s	466.50 min
Programmable one-shot	1.63 μ s	427.1 ms	3.26s	466.50 min
Rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Square-wave rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Software triggered strobe	1.63 μ s	427.1 ms	3.26s	466.50 min
Hardware triggered strobe	1.63 μ s	427.1 ms	3.26s	466.50 min
Event counter	—	2.46 MHz	—	—

Interfaces

MULTIBUS — All signals TTL compatible

ISBX BUS — All signals TTL compatible

PARALLEL I/O — All signals TTL compatible

SERIAL I/O — RS232C compatible, configurable as a data set or data terminal

TIMER — All signals TTL compatible

INTERRUPT REQUESTS — All TTL compatible

Connectors

Interface	Double-Sided Pins (qty)	Centers (in.)	Mating Connectors
MULTIBUS™ System	86	0.156	Viking 3KH43/9AMK12 Wire Wrap
iSBX™ Bus			
8-Bit Data	36	0.1	iSBX 960-5
16-Bit Data	44	0.1	iSBX 961-5
Parallel I/O (2)	50	0.1	3M 3415-000 Flat or TI H312125 Pins
Serial I/O	26	0.1	3M 3462-0001 Flat or AMP 88106-1 Flat

Line Drivers and Terminators

I/O DRIVERS — The following line drivers are all compatible with the I/O driver sockets on the iSBC 86/05 board

Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

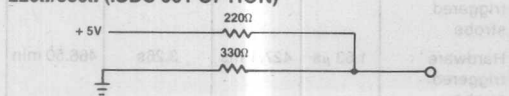
NOTES:

I = inverting; NI = non-inverting; OC = open collector.

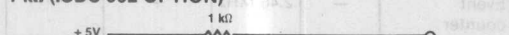
Port 1 of the 8255A has 20 mA totem-pole bidirectional drivers and 1 k Ω terminators

I/O TERMINATORS — 220 Ω /330 Ω divider or 1 k Ω pullup

220 Ω /330 Ω (iSBC 901 OPTION)



1 k Ω (iSBC 902 OPTION)



MULTIBUS Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-State	50
Address	Tri-State	50
Commands	Tri-State	32
Bus Control	Open Collector	20

Physical Characteristics

WIDTH — 12.00 in. (30.48 cm)

HEIGHT — 6.75 in. (17.15 cm)

DEPTH — 0.70 in. (1.78 cm)

WEIGHT — 14 oz (388 gm)

Electrical Characteristics

DC POWER REQUIREMENTS

Configuration	Current Requirements (All Voltages $\pm 5\%$)		
	+ 5V	+ 12V	- 12V
Without EPROM ¹	4.7A	25 mA	23 mA
RAM only ²	120 mA		
With 8K EPROM ³ (using 2716)	5.0A	25 mA	23 mA
With 16K EPROM ³ (using 2732A)	4.9A	25 mA	23 mA
With 32K EPROM ³ (using 2764)	4.9A	25 mA	23 mA

NOTES:

- Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators.
- RAM chips powered via auxiliary power bus in power-down mode.
- Includes power required for 4 ROM/EPROM chips, and I/O terminators installed for 16 I/O lines; all terminator inputs low.

Environmental Characteristics

OPERATING TEMPERATURE — 0°C to 55°C

RELATIVE HUMIDITY — to 90% (without condensation)

Reference Manual

143153-001 — iSBC 86/05 Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

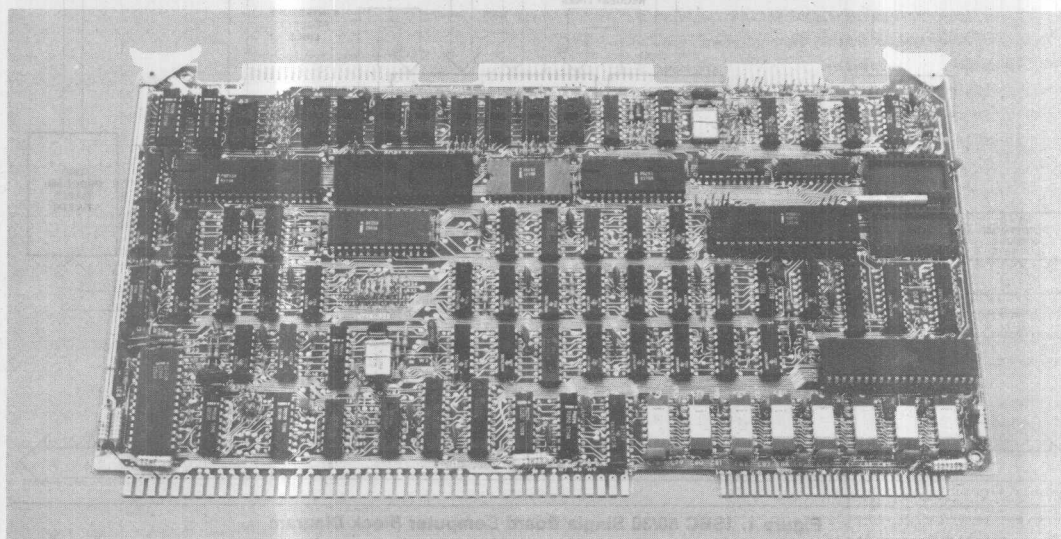
Part Number	Description
SBC 86/05	16-bit Single Board Computer with 8K bytes RAM



iSBC™ 80/30 (or pSBC 80/30*) SINGLE BOARD COMPUTER

- 8085A CPU used as central processing unit
- 16K bytes of dual port dynamic read/write memory with on-board refresh
- Sockets for up to 8K bytes of read only memory
- Sockets for 8041A/8741A Universal Peripheral Interface and interchangeable line drivers and line terminators
- 24 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators
- Programmable synchronous/asynchronous RS232C compatible serial interface with fully software selectable baud rate generation
- Full MULTIBUS® control logic allowing up to 16 masters to share the system
- 12 levels of programmable interrupt control
- Two programmable 16-bit BCD or binary counters
- Auxiliary power bus, memory protect, and power-fail interrupt control logic for RAM battery backup
- Compatible with optional iSBC 80 CPU, memory, and I/O expansion boards

The iSBC 80/30 Single Board Computer is a member of Intel's complete line of OEM computer systems which take full advantage of Intel's LSI technology to provide economical self-contained computer-based solutions for OEM applications. The iSBC 80/30 is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, read/write memory, nonvolatile read only memory, universal peripheral interface capability, I/O ports and drivers, serial communications interface, priority interrupt logic, programmable timers, MULTIBUS control logic, and bus expansion drivers all reside on the board.



*Same product, manufactured by Intel Puerto Rico, Inc.

FUNCTIONAL DESCRIPTION

Central Processing Unit

Intel's powerful 8-bit n-channel 8085A CPU, fabricated on a single LSI chip, is the central processor for the iSBC 80/30. The 8085A CPU is directly software compatible with the Intel 8080A CPU. The 8085A contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators. The minimum instruction execution time is 1.45 microseconds. The 8085A CPU has a 16-bit program counter. An external stack, located within any portion of iSBC 80/30 read/write memory, may be used as a last-in/first-out storage area for the contents of the program counter, flags, accumulator, and all of the six general purpose registers. A 16-bit stack pointer controls the addressing of this external stack. This stack provides subroutine nesting bounded only by memory size.

Bus Structure

The iSBC 80/30 has an internal bus for all on-board memory and I/O operations and a system bus (i.e., the MULTIBUS) for all external memory and I/O operations. Hence, local (on-board) operations do not tie up the system bus, and allow true parallel processing when several bus masters (i.e., DMA devices, other single board computers) are used in a multimaster scheme. A block diagram of the iSBC 80/30 functional components is shown in Figure 1.

RAM Capacity

The iSBC 80/30 contains 16K bytes of dynamic read/write memory using Intel 2117 RAMs. All RAM read and write operations are performed at maximum processor speed. Power for the on-board RAM may be provided on an auxiliary power bus, and memory protect logic is included for RAM battery backup requirements. The iSBC 80/30 contains a dual port controller, which provides dual port capability for the on-board RAM memory. RAM accesses may occur from either the iSBC 80/30 or from any other bus master interfaced via the

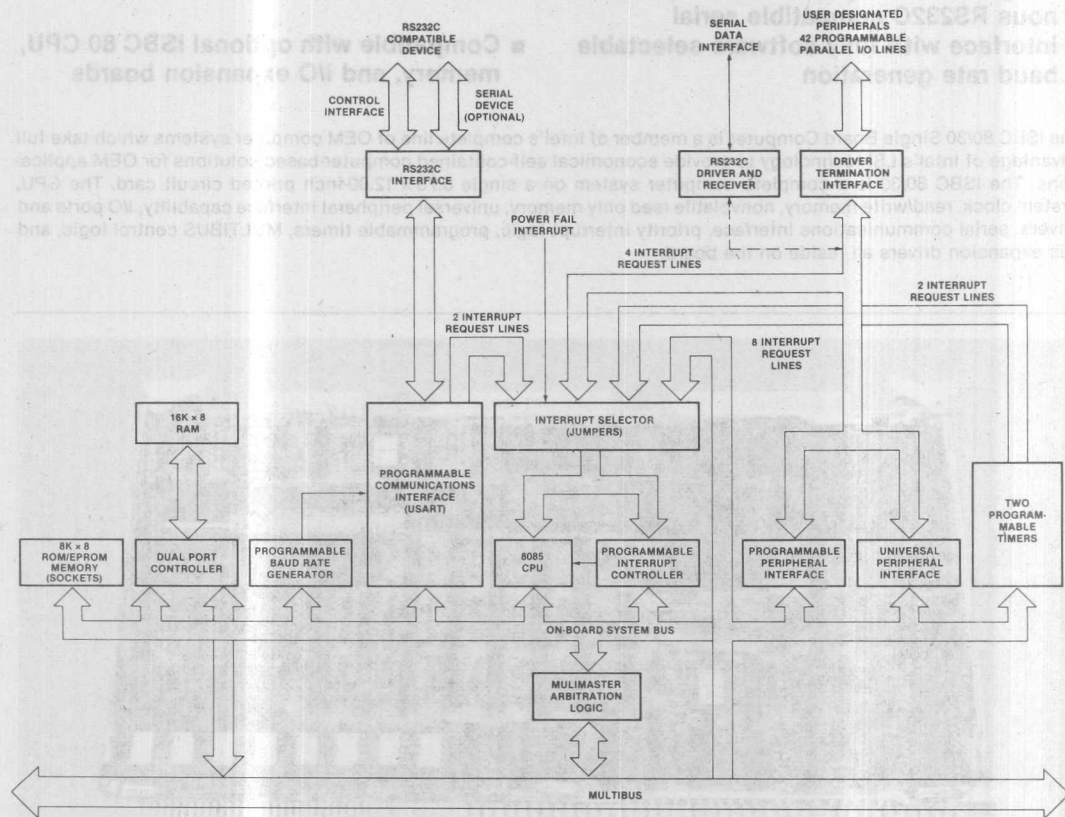


Figure 1. iSBC 80/30 Single Board Computer Block Diagram

MULTIBUS. Since on-board RAM accesses do not require the MULTIBUS, the bus is available for any other concurrent operations (e.g., DMA data transfers) requiring the use of the MULTIBUS. Dynamic RAM refresh is accomplished automatically by the iSBC 80/30 for accesses originating from either the CPU or via the MULTIBUS. Memory space assignment can be selected independently for on-board and MULTIBUS RAM accesses. The on-board RAM, as seen by the 8085A CPU, may be placed anywhere within the 0- to 64K-address space. The iSBC 80/30 provides extended addressing jumpers to allow the on-board RAM to reside within a one megabyte address space when accessed via the MULTIBUS. In addition, jumper options are provided which allow the user to reserve 8K- and 16K-byte segments of on-board RAM for use by the 8085A CPU only. This reserved RAM space is not accessible via the MULTIBUS and does not occupy any system address space.

EPROM/ROM Capacity

Sockets for up to 8K bytes of nonvolatile read only memory are provided on the iSBC 80/30 board. Read only memory may be added in 1K-byte increments up to a maximum of 2K bytes using Intel 2708 or 2758 erasable and electrically reprogrammable ROMs (EPROMs); in 2K-byte increments up to a maximum of 4K bytes using Intel 2716 EPROMs; or in 4K-byte increments up to 8K bytes maximum using Intel 2732 EPROMs. All on-board EPROM/ROM operations are performed at maximum processor speed.

Parallel I/O Interface

The iSBC 80/30 contains 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral Interface. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports indicated in Table 1. Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibil-

ity of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector that mates with flat, woven, or round cable.

Universal Peripheral Interface (UPI)

The iSBC 80/30 provides sockets for a user supplied Intel 8041A/8741A Universal Peripheral Interface (UPI) chip and the associated line drivers and terminators for the UPI's I/O ports. The 8041A/8741A is a single chip microcomputer containing a CPU, 1K bytes of ROM (8041A) or EPROM (8741A), 64 bytes of RAM, 18 programmable I/O lines, and an 8-bit timer. Special interface registers included in the chip allow the 8041A to function as a slave processor to the iSBC 80/30's 8085A CPU. The UPI allows the user to specify algorithms for controlling user peripherals directly in the chip, thereby relieving the 8085A for other system functions. The iSBC 80/30 provides an RS232C driver and an RS232C receiver for optional connection to the 8041A/8741A in applications where the UPI is programmed to handle simple serial interfaces. For additional information, including 8041A/8741A instructions, refer to the UPI-41A User's Manual and application note AP-41.

Serial I/O

A programmable communications interface using the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 80/30. A software selectable baud rate generator provides the USART with all common communication frequencies. The USART can be programmed by the system software to select the desired asynchronous or synchronous serial data transmission technique (including IBM By-Sync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation				Control	
		Unidirectional					
		Input		Output			Bidirectional
		Unlatched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X		
2	8	X	X	X	X		
3	4	X		X		X ¹	
	4	X		X		X ¹	

Note

1. Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

and asynchronous and synchronous modems. The RS232C command lines, serial data lines, and signal ground line are brought out to a 26-pin edge connector that mates with RS232C compatible flat or round cable.

Multimaster Capability

The iSBC 80/30 is a full computer on a single board with resources capable of supporting a great variety of OEM system requirements. For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication over the system bus), the iSBC 80/30 provides full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 80/30's or other bus masters to share the system bus in serial (daisy chain) priority fashion, and up to 16 masters to share the MULTIBUS with the addition of an external priority network. The MULTIBUS arbitration logic operates synchronously with a MULTIBUS clock (provided by the iSBC 80/30 or optionally connected directly to the MULTIBUS clock) while data is transferred via a handshake between the master and slave modules. This allows different speed controllers to share resources on the same bus, and transfers via the bus proceed asynchronously. Thus, transfer speed is dependent on transmitting and receiving devices only. This design prevents slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed direct memory access (DMA) operations, and high speed peripheral control, but are by no means limited to these three.

Programmable Timers

The iSBC 80/30 provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller, to the I/O line drivers associated with the 8255A Programmable Peripheral Interface, and to the 8041A/8741A Universal Programmable Interface, or may be routed as inputs to the 8255A and 8041A/8741A chips. The gate/trigger inputs may be routed to I/O terminators associated with the 8255A or as output connections from the 8255A. The third interval timer in the 8253 provides the programmable baud rate generator for the iSBC 80/30 RS232C USART serial port. In utilizing the iSBC 80/30, the systems designer simply configures, via software, each timer independently to meet system requirements.

counters select the desired function. Seven functions are available, as shown in Table 2. The contents of each counter may be read at any time during system operation with simple read operations for event counting applications, and special commands are included so that the contents of each counter can be read "on the fly".

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low-going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge on counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counting "window" has been enabled or an interrupt may be generated after N events occur in the system.

Interrupt Capability

The iSBC 80/30 provides vectoring for 12 interrupt levels. Four of these levels are handled directly by the interrupt processing capability of the 8085A CPU and represent the four highest priority interrupts of the iSBC 80/30. Requests are routed to the 8085A interrupt inputs, TRAP, RST 7.5, RST 6.5, and RST 5.5 (in decreasing order of priority) and each input generates a unique memory address (TRAP: 24H; RST 7.5: 3CH; RST 6.5: 34H; and RST 5.5: 2CH). An 8085A jump instruction at each of these addresses then provides linkage to interrupt ser-

vice routines located independently anywhere in memory. All interrupt inputs with the exception of the trap interrupt may be masked via software. The trap interrupt should be used for conditions such as power-down sequences which require immediate attention by the 8085A CPU. The Intel 8259A Programmable Interrupt Controller (PIC) provides vectoring for the next eight interrupt levels. As shown in Table 3, a selection of four priority processing modes is available to the systems designer for use in designing request processing configurations to match system requirements. Operating mode and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from the programmable parallel and serial I/O interfaces, the programmable timers, the system bus, or directly from peripheral equipment. The PIC then determines which of the incoming requests is of the highest priority, determines whether this request is of higher priority than the level currently being serviced, and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. The PIC generates a unique memory address for each interrupt level. These addresses are equally spaced at intervals of 4 or 8 (software selectable) bytes. This 32- or 64-byte block may be located to begin at any 32- or 64-byte boundary in the 65,536-byte memory space. A single 8085A jump instruction at each of these addresses then provides linkage to locate each interrupt service routine independently anywhere in memory.

Table 3. Programmable Interrupt Modes

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

Interrupt Request Generation — Interrupt requests may originate from 18 sources. Two jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface when a byte of information is ready to be transferred to the CPU (i.e., input buffer is full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). Two jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU (i.e., receive channel buffer is full), or a character is ready to be transmitted (i.e., transmit channel data buffer is empty). A jumper selectable request can be generated by each of the programmable timers and by the universal peripheral interface, eight additional interrupt request lines

are available to the user for direct interface to user designated peripheral devices via the system bus, and two interrupt request lines may be jumper routed directly from peripherals via the parallel I/O driver/terminator section.

Power-Fail Control

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the ISBC 635 Power Supply or equivalent.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added by using Intel MULTIBUS compatible expansion boards. High speed integer and floating point arithmetic capabilities may be added by using the ISBC 310A High Speed Mathematics Unit. Memory may be expanded to 65,536 bytes by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers as subsystems. Modular expandable backplanes and card-cages are available to support multi-board systems.

Real-Time Software

Intel's iRMX 80 Real-Time Multi-Tasking Executive software, specifically designed for Intel ISBC 80 single board computers, provides the capability to monitor and control multiple asynchronous external events. The iRMX 80 executive, which synchronizes and controls the execution of multiple tasks, is provided as a linkable and relocatable module requiring only 2K bytes of memory space. Optional linkable and relocatable modules for teletypewriter and CRT control, diskette file system, high speed math unit, and analog subsystems are also available.

System Development Capability

The development cycle of ISBC 80/30-based products may be significantly reduced using Intel's system development tools available today. For those not requiring hardware emulation capability, Intel provides a new low cost microcomputer development system. The iPDS, Personal Development System, provides low cost system development for the ISBC 80/30 board, while at the same time providing personal computer capability for the engineer. The Intellec Series II family of compatible microcomputer development systems provides a range of capability from a low cost disk-based edit debug workstation to a high performance, fully compatible hard-disk-based software development system. A unique in-circuit emulator (ICE-85A) option provides the capability of developing and debugging software directly on the ISBC 80/30 board.

Programming Capability

PL/M-80 — Intel's high level programming language, PL/M, is also available as a resident Intellec microcomputer development system option. PL/M provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or

allocate memory. PL/M programs can be written in a much shorter time than assembly language programs for a given application.

FORTTRAN-80 — For applications requiring computational and formatted I/O capabilities, the high level FORTRAN-80 programming language is also available as a resident option of the intellec system. FORTRAN-80 meets and exceeds the ANS FORTRAN 77 subset language specification. The FORTRAN-80 compiler produces relocatable object code that may be easily linked with other FORTRAN-80, PL/M, or assembly language program modules. This gives the user wide flexibility in developing software by using the best software tool for a particular functional module within the user's application.

SPECIFICATIONS

Word Size

Instruction — 8, 16, or 24 bits

Data — 8 bits

Cycle Time

Basic Instruction Cycle — 1.45 μ s

Note

Basic instruction cycle is defined as the fastest instruction (i.e., four clock cycles).

Memory Addressing

On-Board ROM/EPROM — 0-07FF (using 2708 or 2758 EPROMs); 0-0FFF (using 2716 EPROMs); 0-1FFF (using 2716 EPROMs; 0-1FFF (using 2732 EPROMs).

On-Board RAM — 16K bytes of dual port RAM starting on a 16K boundary. One or two 8K-byte segments may be reserved for CPU use only.

Memory Capacity

On-Board Read Only Memory — 8K bytes (sockets only)

On-Board RAM — 16K bytes

Off-Board Expansion — Up to 65,536 bytes in user specified combinations of RAM, ROM, and EPROM

Note

Read only memory may be added in 1K, 2K, or 4K-byte increments.

I/O Addressing

On-Board Programmable I/O (see Table 1)

Port	8255A				8041A/8741A				USART	
	1	2	3	Control	Data	Control	Data	Control	Data	Control
Address	E8	E9	EA	EB	E4 or E6	E5 or E7	EC	ED		

I/O Capacity

Parallel — 42 programmable lines using one 8255A (24 I/O lines) and an optional 8041A/8741A (18 I/O lines)

Serial — 2 programmable lines using one 8251A and an optional 8041A/8741A programmed for serial operation

Note:

For additional information on the 8041A/8741A refer to the UPI-41 User's Manual (Publication 9800504).

BASIC-80 — A high level language interpreter with extended disk capabilities which operates under the iRMX 80 Real-Time Multi-tasking Executive and translates BASIC-80 source programs into an internally executable form. This language interpreter, provided as a set of linkable object modules, is ideally suited to the OEM who requires a pass thru programming language. The BASIC-80 programs may be created, stored and interpreted on the ISBC 80-based system. The BASIC-80 language has a rich complement of statements, functions, and commands to program applications requiring a full range of 1) string manipulation and disk I/O for data processing, 2) single and double precision floating point and array handling for numeric analysis, or 3) port I/O with mask operations controlled through bit-wise Boolean logical operators.

Serial Communications Characteristics

Synchronous — 5—8 bit characters; internal or external character synchronization; automatic sync insertion.

Asynchronous — 5—8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection.

Baud Rates

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)			
	Synchronous	Asynchronous		
		+ 16	+ 64	
153.6	—	9600	2400	
76.8	—	4800	1200	
38.4	38400	2400	600	
19.2	19200	1200	300	
9.6	9600	600	150	
4.8	4800	300	75	
2.4	2400	150	—	
1.76	1760	110	—	

Note

Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register (8253 Timer 2).

Interrupts

Addresses for 8259A Registers (Hex notation, I/O address space)

DA Interrupt request register

DA In-service register

DB Mask register

DA Command register

DB Block address register

DA Status (polling register)

Note

Several registers have the same physical address; sequence of access and one data bit of control word determine which register will respond.

Interrupt Levels routed to 8085A CPU automatically vector the processor to unique memory locations:

Interrupt Input	Memory Address	Priority	Type
TRAP	24	Highest	Non-maskable
RST 7.5	3C	<div style="text-align: center;"> ↑ ↓ </div>	Maskable
RST 6.5	34		Maskable
RST 5.5	2C		Maskable

Timers

Register Addresses (Hex notation, I/O address space)

DF Control register

DC Timer 0

DD Timer 1

DE Timer 2

Note

Timer counts loaded as two sequential output operations to same address, as given.

Input Frequencies

Reference: 2.46 MHz \pm 0.1% (0.041 μ s period, nominal);
1.23 MHz \pm 0.1% (0.81 μ s period, nominal); or 153.60 kHz
 \pm 0.1% (6.51 μ s period nominal).

Note

Above frequencies are user selectable

Event Rate: 2.46 MHz max

Note

Maximum rate for external events in event counter function.

Output Frequencies/Timing Intervals

Function	Single Timer/Counter		Dual Timer/Counter (Two Timers Cascaded)	
	Min	Max	Min	Max
Real-time interrupt	1.63 μ s	427.1 ms	3.26 μ s	466.50 min
Programmable one-shot	1.63 μ s	427.1 ms	3.26 μ s	466.50 min
Rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Square-wave rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Software triggered strobe	1.63 μ s	427.1 ms	3.26 μ s	466.50 min
Hardware triggered strobe	1.63 μ s	427.1 ms	3.26 μ s	466.50 min

Interfaces

MULTIBUS — All signals TTL compatible

Parallel I/O — All signals TTL compatible

Interrupt Requests — All TTL compatible

Timer — All signals TTL compatible

Serial I/O — RS232C compatible, data set configuration

System Clock (8085A CPU)

2.76 MHz \pm 0.1%

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM for systems requiring battery backup of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

Connectors

Interface	Pins (qty)	Centers (in.)	Mating Connectors
Bus	86	0.156	Viking 2KH43/9AMK12
Parallel I/O	50	0.1	3M 3415-000
Serial I/O	26	0.1	3M 3462-000

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power-down sequences.

Line Drivers and Terminators

I/O Drivers — The following line drivers are all compatible with the I/O driver sockets on the iSBC 80/30.

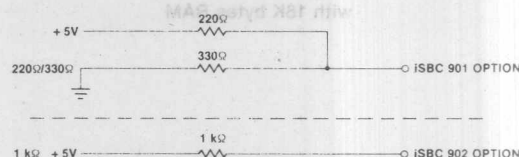
Driver	Characteristic	Sink Current (mA)
7438	I.O.C	48
7437	I	48
7432	NI	16
7426	I.O.C	16
7409	NI.O.C	16
7408	NI	16
7403	I.O.C	16
7400	I	16

Note

I = inverting; NI = non-inverting; OC = open collector.

Port 1 of the 8255A has 20 mA totem-pole bidirectional drivers and 1 k Ω terminators.

I/O Terminators — 220 Ω /330 Ω divider or 1 k Ω pullup



Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-state	50
Address	Tri-state	50
Commands	Tri-state	32

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.50 in. (1.27 cm)

Weight — 18 oz. (509.6 gm)

Configuration	Current Requirements			
	VCC = +5V ± 5%(max)	VDD = +12V ± 5%(max)	VBB = -5V ± 5%(max)	VAA = -12V ± 5%(max)
Without EPROM ¹	I _{CC} = 3.5A	I _{DD} = 220 mA	I _{BB} = —	I _{AA} = 50 mA
With 8041/8741 ²	3.6A	220 mA	—	50 mA
RAM only ³	350 mA	20 mA	2.5 mA	—
With ISBC 530 ⁴	3.5A	320 mA	—	150 mA
With 2K EPROM ⁵ (using 8708)	4.4A	350 mA	95 mA	40 mA
With 2K EPROM ⁵ (using 2758)	4.6A	220 mA	—	50 mA
With 4K EPROM ⁵ (using 2716)	4.6A	220 mA	—	50 mA
With 8K EPROM ⁵ (using 2332)	4.6A	220 mA	—	50 mA

Notes

- Does not include power required for optional EPROM/ROM, 8041A/8741A I/O drivers, and I/O terminators.
- Does not include power required for optional EPROM/ROM, I/O drivers and I/O terminators.
- RAM chips powered via auxiliary power bus
- Does not include power required for optional EPROM/ROM, 8041A/8741A I/O drivers, and I/O terminators. Power for ISBC 530 is supplied through the serial port connector.
- Includes power required for two EPROM/ROM chips, 8041A/8741A and 2200/3300 input terminators installed for 34 I/O lines; all terminator inputs low.

ORDERING INFORMATION

Part Number	Description
SBC 80/30	Single Board Computer with 16K bytes RAM

Reference Manual

9800611B — ISBC 80/30 Single Board Computer Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

Function	Single Time Counter		Dual Time Counter (Two Times Cascaded)	
	Min	Max	Min	Max
Read time	1.03 μ s	427.7 ms	3.20 μ s	468.50 min
Interrupt	1.03 μ s	427.7 ms	3.20 μ s	468.50 min
Program transfer	1.03 μ s	427.7 ms	3.20 μ s	468.50 min
Rate generator	2.04 Hz	870.8 kHz	0.00008 Hz	308.8 kHz
Sound wave	2.04 Hz	870.8 kHz	0.00008 Hz	308.8 kHz
Auto generator	1.03 μ s	427.7 ms	3.20 μ s	468.50 min
Stop	1.03 μ s	427.7 ms	3.20 μ s	468.50 min
Hardware interrupt	1.03 μ s	427.7 ms	3.20 μ s	468.50 min
Stop	1.03 μ s	427.7 ms	3.20 μ s	468.50 min

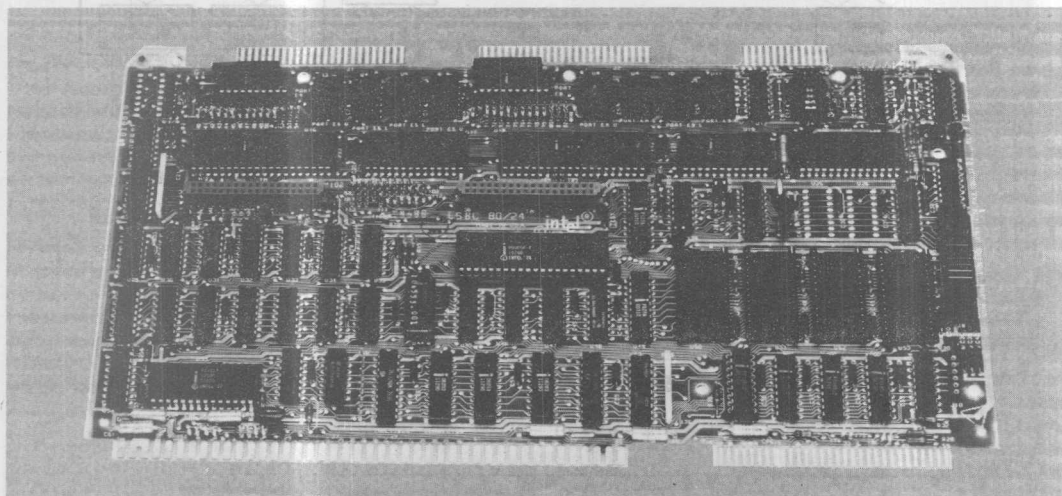
Bus Drivers		Function	
Characteristics	Static Current (mA)	Characteristics	Static Current (mA)
Tri-state	50	Tri-state	50
Tri-state	50	Tri-state	50
Tri-state	50	Tri-state	50



iSBC™ 80/24 (or pSBC 80/24*) SINGLE BOARD COMPUTER

- Upward compatible with iSBC 80/20-4 Single Board Computer
- 8085A-2 CPU operating at 4.8 or 2.4 MHz
- Two iSBX™ bus connectors for iSBX MULTIMODULE™ board expansion
- 4K bytes of static read/write memory expandable on-board to 8K bytes using the iSBC 301 MULTIMODULE Board
- Sockets for up to 32K bytes of read only memory
- 48 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators
- Programmable synchronous/asynchronous RS232C compatible serial interface with software selectable baud rates
- Full MULTIBUS® control logic for multimaster configurations and system expansion
- Two programmable 16-bit BCD or binary timers/event counters
- 12 levels of programmable interrupt control
- Auxiliary power bus, memory protect, and power-fail interrupt control logic provided for battery backup RAM requirements

The Intel® iSBC 80/24 Single Board Computer is a member of Intel's complete line of OEM microcomputer systems which take full advantage of Intel's LSI technology to provide economical, self-contained computer-based solutions for OEM applications. The iSBC 80/24 board is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, iSBX bus interface, read/write memory, read only memory sockets, I/O ports and drivers, serial communications interface, priority interrupt logic, and programmable timers all reside on the board. Full MULTIBUS interface logic is included to offer compatibility with the Intel OEM Microcomputer Systems family of Single Board Computers, expansion memory options, digital and analog I/O expansion boards, and peripheral and communications controllers.



*Same product, manufactured by Intel Puerto Rico, Inc.

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Index, Intel, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, iRMX, UPI, uScope, Promware, MCS, ICE, iSBC, iSBX, MULTIMODULE and ICS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1980, 1981

FUNCTIONAL DESCRIPTION

Central Processing Unit

Intel's powerful 8-bit N-channel 8085A-2 CPU fabricated on a single LSI chip, is the central processor for the iSBC 80/24 board operating at either 4.8 or 2.4 MHz (jumper selectable). The 8085A-2 CPU is directly software compatible with the Intel 8080A CPU. The 8085A-2 contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing single and double precision operators. Minimum instruction execution time is 826 nanoseconds. A block diagram of the iSBC 80/24 functional components is shown in Figure 1.

MULTIMODULE Board Expansion

The new iSBX bus interface brings an entirely new dimension to system design offering incremental on-board expansion at minimal cost. Two iSBX bus MULTIMODULE connectors are provided for plug-in expansion of any iSBX MULTIMODULE board. The iSBX MULTIMODULE concept provides the ability to adapt quickly to new technology, the economy of buying only what is needed, and the ready availability of a spectrum of functions for greater application potential. iSBX boards are

available to provide expansion equivalent to the I/O available on the iSBC 80/24 board or the user may configure entirely new functionality, such as math, directly on board. The iSBX 350 Parallel I/O MULTIMODULE board provides 24 I/O lines using an 8255A Programmable Peripheral Interface. Therefore, two iSBX 350 modules together with the iSBC 80/24 board may offer 96 lines of programmable I/O. Alternately, a serial port may be added using the iSBX 351 Serial I/O MULTIMODULE board and math may be configured on-board with the iSBX 332 Floating Point Math or iSBX 331 Fixed/Floating Point Math MULTIMODULE board. Future iSBX products are also planned. The iSBX MULTIMODULE board is a logical extension of the on-board programmable I/O and is accessed by the iSBC 80/24 single board computer as common I/O port locations. The iSBX board is coupled directly to the 8085A-2 CPU and therefore becomes an integral element of the iSBC 80/24 single board computer providing optimum performance. In addition, RAM memory capacity may be expanded to 8K bytes using the iSBC 301 4K Byte RAM MULTIMODULE board. All MULTIMODULE boards ranging from the iSBC 301 module to the iSBX modules offer incremental expansion, optimum performance, and minimal cost.

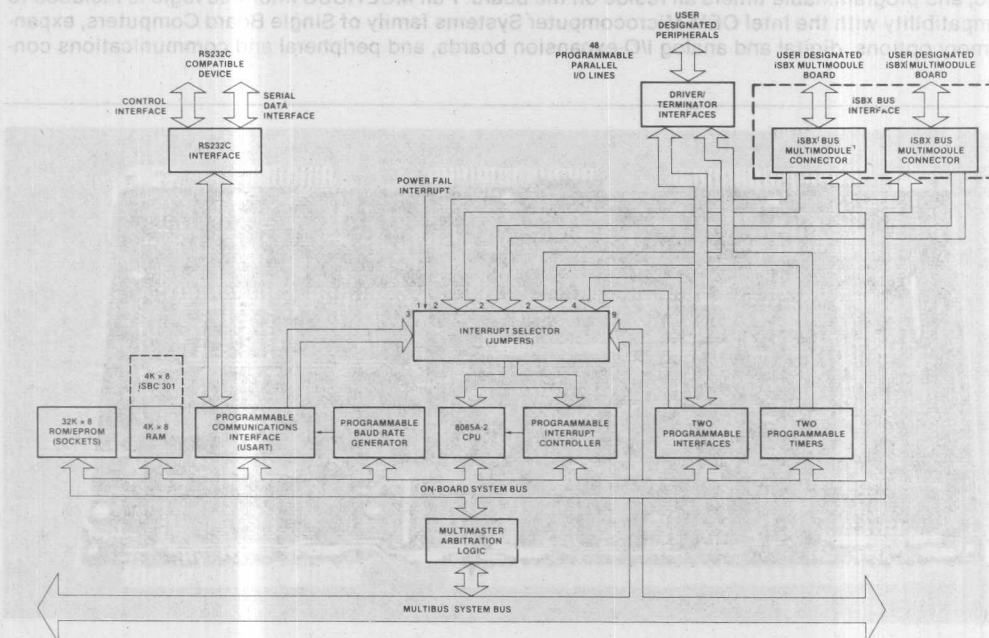


Figure 1. iSBC 80/24 Single Board Computer Block Diagram

Memory Addressing

The 8085A-2 has a 16-bit program counter which allows direct addressing of up to 64K bytes of memory. An external stack, located within any portion of read/write memory, may be used as a last-in/first-out storage area for the contents of the program counter, flags, accumulator, and all of the six general purpose registers. A 16-bit stack pointer controls the addressing of this external stack. This stack provides subroutine nesting bounded only by memory size.

Memory Capacity

The iSBC 80/24 board contains 4K bytes of static read/write memory using Intel 8185-2 RAMs. In addition, the on-board RAM capacity may be expanded to 8K bytes with the iSBC 301 4K byte RAM MULTIMODULE board. All RAM read and write operations are performed at maximum processor speed. Power for the on-board RAM may be provided on an auxiliary power bus, and memory protect logic is included for RAM battery backup requirements.

Four sockets are provided for up to 32K bytes of nonvolatile read only memory on the iSBC 80/24 board. EPROM may be added in 1K byte increments up to 4K bytes (using Intel 2708 or 2758); in 2K byte increments up to 8K bytes (using Intel 2716); in 4K byte increments up to 16K bytes (using Intel 2732); or in 8K byte increments up to 32K bytes (using Intel 2764).

Parallel I/O Interface

The iSBC 80/24 board contains 48 programmable parallel I/O lines implemented using two Intel 8255A Programmable Peripheral Interfaces. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports as indicated in Table 1. Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 48 programmable I/O lines and signal ground lines are brought out to two 50-pin edge connectors that mate with flat, woven, or round cables.

Serial I/O Interface

A programmable communications interface using the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 80/24 board. A software selectable baud rate generator provides the USART with all common communication frequencies. The USART can be programmed by the system software to select the desired asynchronous or synchronous serial data transmission technique (including IBM

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation					
		Unidirectional				Bidirectional	Control
		Input		Output			
		Unlatched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	4	X		X			X ¹
	4	X		X			X ¹
4	8	X	X	X	X	X	
5	8	X	X	X	X		
6	4	X		X			X ²
	4	X		X			X ²

NOTES:

- Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.
- Part of port 6 must be used as a control port when either port 4 or port 5 are used as a latched and strobed input or a latched and strobed output port or port 4 is used as a bidirectional port.

Bi-Sync). The mode of operation (i.e. synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The RS232C compatible interface, in conjunction with the USART, provides a direct interface to RS232C compatible terminals, cassettes, and asynchronous and synchronous modems. The RS232C command lines, serial data lines, and signal ground line are brought out to a 26-pin edge connector that mates with RS232C compatible flat or round cable.

Multimaster Capability

The iSBC 80/24 board is a full computer on a single board with resources capable of supporting a large variety of OEM system requirements. For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e. several CPUs and/or controllers logically sharing system tasks through communication over the system bus), the iSBC 80/24 board provides full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 80/24 boards or other bus masters to share the system bus in serial (daisy chain) priority fashion, and up to 16 masters to share the MULTIBUS system bus with the addition of an external priority network. The MULTIBUS arbitration logic operates synchronously with a MULTIBUS clock (provided by the iSBC 80/24 board or optionally connected directly to the MULTIBUS clock) while data is transferred via a handshake between the master and slave modules. This allows different speed controllers to share resources on the same bus since transfers via the bus proceed asynchronously. Thus, transfer speed is dependent on transmitting and receiving devices only. This design provides slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed direct memory access (DMA) operations, and high speed peripheral control, but are by no means limited to these three.

Programmable Timers

The iSBC 80/24 board provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable In-

terval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller, to the I/O line drivers associated with the 8255A Programmable Peripheral Interface, or may be routed as inputs to the 8255A chip. The gate/trigger inputs may be routed to I/O terminators associated with the 8255A or as output connections from the 8255A. The third interval timer in the 8253 provides the programmable baud rate generator for the RS232C USART serial port. In utilizing the iSBC 80/24 board, the systems designer simply configures, via software, each timer independently to meet system require-

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low-going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge on counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counting "window" has been enabled or an interrupt may be generated after N events occur in the system.

ments. Whenever a given time delay or count is needed, software commands to the programmable timers/event counters select the desired function. Seven functions are available, as shown in Table 2. The contents of each counter may be read at any time during system operation with simple read operations for event counting applications, and special commands are included so that the contents of each counter can be read "on the fly".

Interrupt Capability

The iSBC 80/24 board provides vectoring for 12 interrupt levels. Four of these levels are handled directly by the interrupt processing capability of the 8085A-2 CPU and represent the four highest priority interrupts of the iSBC 80/24 board. Requests are routed to the 8085A-2 interrupt inputs—TRAP, RST 7.5, RST 6.5, and RST 5.5 (in decreasing order of priority), each of which generates a call instruction to a unique address (TRAP: 24H; RST 7.5: 3CH; RST 6.5: 34H; and RST 5.5: 2CH). An 8085A-2 JMP instruction at each of these addresses then provides linkage to interrupt service routines located independently anywhere in memory. All interrupt inputs with the exception of the trap interrupt may be masked via software. The trap interrupt should be used for conditions such as power-down sequences which require immediate attention by the 8085A-2 CPU. The Intel 8259A Programmable Interrupt Controller (PIC) provides vectoring for the next eight interrupt levels. As shown in Table 3, a selection of four priority processing modes is available to the systems designer for use in designing request processing configurations to match system requirements. Operating mode and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from the programmable parallel and serial I/O interfaces, the programmable timers, the system bus, iSBX bus, or directly from peripheral equipment. The PIC then determines which of the incoming requests is of the highest priority, determines whether this request is of higher priority than the level currently being serviced, and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. The PIC generates a unique memory address for each interrupt level. These addresses are equally spaced at intervals of 4 or 8 (software selectable) bytes. This 32 or 64-byte block may be located to begin at any 32 or 64-byte boundary in the 65,536-byte memory space. A single 8085A-2 JMP instruction at each of these addresses then provides linkage to locate each interrupt service routine independently anywhere in memory.

Table 3. Programmable Interrupt Modes

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Autorotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

Interrupt Request Generation

Interrupt requests may originate from 23 sources. Two jumper selectable interrupt requests can be generated by each iSBX MULTIMODULE board. Two jumper selectable interrupt requests can be automatically generated by each programmable peripheral interface when a byte of information is ready to be transferred to the CPU (i.e., input buffer is full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). Three jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU (i.e., receiver channel buffer is full), a character is ready to be transmitted (i.e., the USART is ready to accept a character from the CPU), or when the transmitter is empty (i.e., the USART has no character to transmit). A jumper selectable request can be generated by each of the programmable timers. Nine interrupt request lines are available to the user for direct interface to user designated peripheral devices via the MULTIBUS system bus. A power-fail signal can also be selected as an interrupt source.

Power-Fail Control

A power-fail interrupt may be detected through the AC-low signal generated by the power supply. This signal may be configured to interrupt the 8085A-2 CPU to initiate an orderly power down instruction sequence.

MULTIBUS System Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS system compatible expansion boards. Memory may be expanded to 65,536 bytes by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capa-

city may be increased by adding digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette or hard disk controllers as subsystems. Expanded communication needs can be handled by communication controllers. Modular expandable backplanes and card cages are available to support multiboard systems.

Real-Time Software

The iRMX™ 80 executive, which contains all major real-time facilities including priority-based system resource allocation, intertask communication and control, interrupt driven control for standard I/O devices, and interrupt handling, occupies 2K bytes of memory which can be stored on-board in EPROM. Optional linkable and relocatable modules for console control (CRT or TTY), disk file system, and analog subsystems are provided with the iRMX 80 package. These facilities eliminate the need for users to design and implement application specific executives, greatly simplifying application design and reducing development time and risk.

System Development Capability

The development cycle of iSBC 80/24-based products may be significantly reduced using Intel's system development tools available today. For those not requiring hardware emulation capability, Intel provides a new low cost microcomputer development system. The iPDS, Personal Development System, provides low cost system development for iSBC 80/24 board, while at the same time providing personal computer capability for the engineer. The Intellec Series II family of compatible microcomputer development systems provides a range of capability from a low cost disk-based edit debug workstation to a high performance, fully compatible hard-disk-based software development system. A unique in-circuit emulator (ICE-85A) option provides the capability of developing and debugging software directly on the iSBC 80/24 board.

SPECIFICATIONS

Word Size

Instruction—8, 16, or 24 bits

Data—8 bits

Cycle Time

Basic Instruction Cycle

826 nsec (4.84 MHz operating frequency)

1.65 μ sec (2.42 MHz operating frequency)

NOTE:

Basic instruction cycle is defined as the fastest instruction (i.e., four clock cycles).

Programming Capability

PL/M-80—Intel's high level system programming language, PL/M, is also available as a resident Intellec microcomputer development system option. PL/M provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M programs can be written in a much shorter time than assembly language programs.

FORTTRAN-80—For applications requiring computational and formatted I/O capabilities, the ANSI 77 standard high level FORTRAN-80 programming language is available as a resident option of the Intellec system. The FORTRAN compiler produces relocatable object code that may be easily linked with PL/M or assembly language program modules. In addition, the iSBC 801 FORTRAN-80 Run-Time Package is a complete, ready-to-use set of linkable object modules which are fully compatible with iRMX 80 systems. The modules, when combined with the FORTRAN-80 coded application, provide the appropriate interfaces to the disk file and terminal I/O of iRMX 80, and to the iSBC 310A Math Unit for applications requiring high speed math.

BASIC-80—A high level language interpreter is available with extended disk capabilities which operates under the iRMX 80 Real-Time Multitasking Executive and translates BASIC-80 source programs into an internally executable form. This language interpreter, provided as a set of linkable object modules, is ideally suited to the OEM who requires a pass through programming language. The BASIC-80 programs may be created, stored, and interpreted on the iSBC 80-based systems using the iSBC 802 BASIC-80 Configurable iRMX 80 Disk-Based Interpreter. The iSBC 802 Interpreter has a complete ready-to-use set of linkable object modules which are fully compatible with Intel's iRMX 80 Real-Time Multitasking Executive Software. The modules provide interfaces to disk file and terminal I/O, software floating point, or interface to other routines provided by the user.

Memory Addressing

On-Board EPROM

0-0FFF using 2708, 2758 (1 wait state)

0-1FFF using 2716 (1 wait state)

0-3FFF using 2732 (1 wait state)

using 2732A (no wait states)

0-7FFF using 2764A (no wait states)

On-Board RAM

3000-3FFF with no RAM expansion

2000-3FFF with optional RAM (iSBC 301 board)

NOTE:

Default configuration—may be reconfigured to top end of any 16K boundary.

Memory Capacity

On-Board EPROM

32K bytes (sockets only)

May be added in 1K (using Intel 2708 or 2758), 2K (using Intel 2716), 4K (using Intel 2732), or 8K (using Intel 2764) byte increments.

On-Board RAM

4K bytes (8K bytes using iSBC 301 4K byte RAM MULTIMODULE Board)

Off-Board Expansion

Up to 64K bytes using user specified combinations of RAM, ROM, and EPROM.

Up to 128K bytes using bank select control via I/O port and 2 jumper options.

May be disabled using PROM ENABLE via I/O port and jumper option, resulting in off-board RAM overlay capability.

I/O Addressing

On-Board Programmable I/O

Device	I/O Address
8255A No. 1	
Port A	E4
Port B	E5
Port C	E6
Control	E7
8255A No. 2	
Port A	E8
Port B	E9
Port C	EA
Control	EB
8251A	
Data	EC, EE
Control	ED, EF
iSBX MULTIMODULE J5	
MCS0	C0-C7
MCS1	C8-CF
iSBX MULTIMODULE J6	
MCS0	F0-F7
MCS1	F8-FF

I/O Capacity

Parallel—48 programmable lines

Serial—1 transmit, 1 receive, 1 SID, 1 SOD

iSBX MULTIMODULE — 2 iSBX MULTIMODULE Boards

Serial Communications Characteristics

Synchronous—5-8 bit characters; internal or external character synchronization; automatic sync insertion

Asynchronous—5-8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detectors

Baud Rates

Output Frequency in kHz	Baud Rate (Hz)	
	Synchronous	Asynchronous
153.6	—	+ 16 + 64 9600 2400
76.8	—	4800 1200
38.4	38400	2400 600
19.2	19200	1200 300
9.6	9600	600 150
4.8	4800	300 75
2.4	2400	150 —
1.76	1760	110 —

NOTE:

Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register.

Register Address (hex notation, I/O address space)

DE Baud rate register

NOTE:

Baud rate factor (16 bits) is loaded as two sequential output operations to same address (DE_H).

Interrupts

Addresses for 8259A Registers (hex notation, I/O address space)

DA or D8 Interrupt request register

DA or D8 In-service register

DB or D9 Mask register

DA or D8 Command register

DB or D9 Block address register

DA or D8 Status (polling register)

NOTE:

Several registers have the same physical address; sequence of access and one data bit of control word determine which register will respond.

Interrupt levels routed to 8085A-2 CPU automatically vector the processor to unique memory locations:

Interrupt Input	Memory Address	Priority	Type
TRAP	24	Highest	Non-maskable
RST 7.5	3C	↓ Lowest	Maskable
RST 6.5	34		Maskable
RST 5.5	2C		Maskable

Register Addresses (hex notation, I/O address space)

DF Control register

DC Timer 0

DD Timer 1

DE Timer 2

NOTE:

Timer counts loaded as two sequential output operations to same address as given.

Input Frequencies

Reference: 1.0752 MHz \pm 0.1% (0.930 μ sec period, nominal)

Event Rate: 1.1 MHz max.

Output Frequencies/Timing Intervals

Function	Single Timer/Counter		Dual Timer/Counter (Two Timers Cascaded)	
	Min.	Max.	Min.	Max.
Real-Time Interrupt	1.86 μ sec	60.948 msec	3.72 μ sec	1,109 hrs
Programmable One-Shot	1.86 μ sec	60.948 msec	3.72 μ sec	1,109 hrs
Rate Generator	16.407 Hz	537.61 kHz	0.00025 Hz	268.81 kHz
Square-Wave Rate Generator	16.407 Hz	537.61 kHz	0.00025 Hz	268.81 kHz
Software Triggered Strobe	1.86 μ sec	60.948 msec	3.72 μ sec	1,109 hrs
Hardware Triggered Strobe	1.86 μ sec	60.948 msec	3.72 μ sec	1,109 hrs

NOTE:
Input frequency to timers is 1.0752 MHz (default configuration).

Interfaces

MULTIBUS—All signals TTL compatible

ISBX Bus—All signals TTL compatible

Parallel I/O—All signals TTL compatible

Serial I/O—RS232C compatible, configurable as a data set or data terminal

Timer—All signals TTL compatible

Interrupt Requests—All TTL compatible

System Clock (8085A-2 CPU)

4.84 or 2.42 MHz \pm 0.1% (jumper selectable)

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM for systems requiring battery

auxiliary RAM power bus is made via jumpers on the board.

Connectors

Interface	Double-Sided Pins (qty)	Centers (in.)	Mating Connectors*
MULTIBUS System Bus	86	0.156	ELFAB BS1562043PBB Viking 2KH43/9AMK12 Soldered PCB Mount EDAC 337086540201 ELFAB BW1562D43PBB EDAC 337086540202 ELFAB BW1562A43PBB Wire Wrap
Auxiliary Bus	60	0.100	EDAC 345060524802 ELFAB BS1020A30PBB EDAC 345060540201 ELFAB BW1020D30PBB Wire Wrap
ISBX Bus (2)	36	0.100	ISBX 960-5
Parallel I/O (2)	50	0.100	3M 3415-001 Flat Crimp GTE Sylvania 6AD01251A1DD Soldered
Serial I/O	26	0.100	AMP 15837151 EDAC 345026520202 PCB Soldered 3M 3462-0001 AMP 88373-5 Flat Crimp

*Note: Connectors compatible with those listed may also be used.

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power-down sequences.

Line Drivers and Terminators

I/O Drivers—The following line drivers and terminators are all compatible with the I/O driver sockets on the iSBC 80/24 Board:

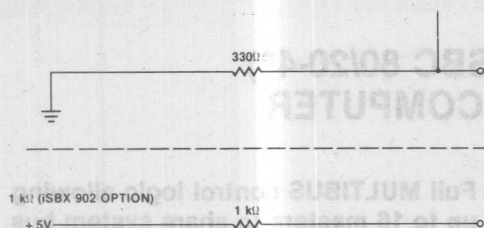
Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

NOTE:

I = inverting; NI = non-inverting; OC = open collector.

Ports E4 and E8 have 32 mA totem-pole drivers and 1K terminators.

I/O Terminators—220 Ω /330 Ω divider or 1 k Ω pullup



Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-State	32
Address	Tri-State	32
Commands	Tri-State	32

Physical Characteristics

Width—12.00 in. (30.48 cm)

Height—6.75 in. (17.15 cm)

Depth—0.50 in. (1.27 cm)

Weight—12.64 oz. (354 gm)

Electrical Characteristics

DC Power Requirements

Configuration	Current Requirements			
	$V_{CC} = +5V$ $\pm 5\%$ (max)	$V_{DD} = +12V$ $\pm 5\%$ (max)	$V_{BB} = -5V$ $\pm 5\%$ (max)	$V_{AA} = -12V$ $\pm 5\%$ (max)
Without EPROM ¹	3.34A	40 mA	—	20 mA
RAM Only ²	0.14A	—	—	—
With ISBC 530 ³	3.34A	140 mA	—	120 mA
With 4K EPROM ⁴ (using 2708)	3.74A	300 mA	180 mA	20 mA
With 4K EPROM ⁴ (using 2758)	4.43A	40 mA	—	20 mA
With 8K EPROM ⁴ (using 2716)	4.43A	40 mA	—	20 mA
With 16K EPROM ⁴ (using 2732)	4.71A	40 mA	—	20 mA
With 32K EPROM ⁴ (using 2764)	4.71A	40 mA	—	20 mA

NOTES:

- Does not include power for optional EPROM, I/O drivers, and I/O terminators.
- RAM chips powered via auxiliary power bus.

ORDERING INFORMATION

Part Number	Description
SBC 80/24	Single Board Computer

The I/O terminators for the ISBC 530 Adapter is supplied via serial port connector.

- Includes power required for four EPROM chips, and I/O terminators installed for 16 I/O lines; all terminator inputs low.

Environmental Characteristics

Operating Temperature—0°C to 55°C

Reference Manual

142648-001—iSBC 80/24 Single Board Computer Hardware Reference Manual (NOT SUPPLIED)

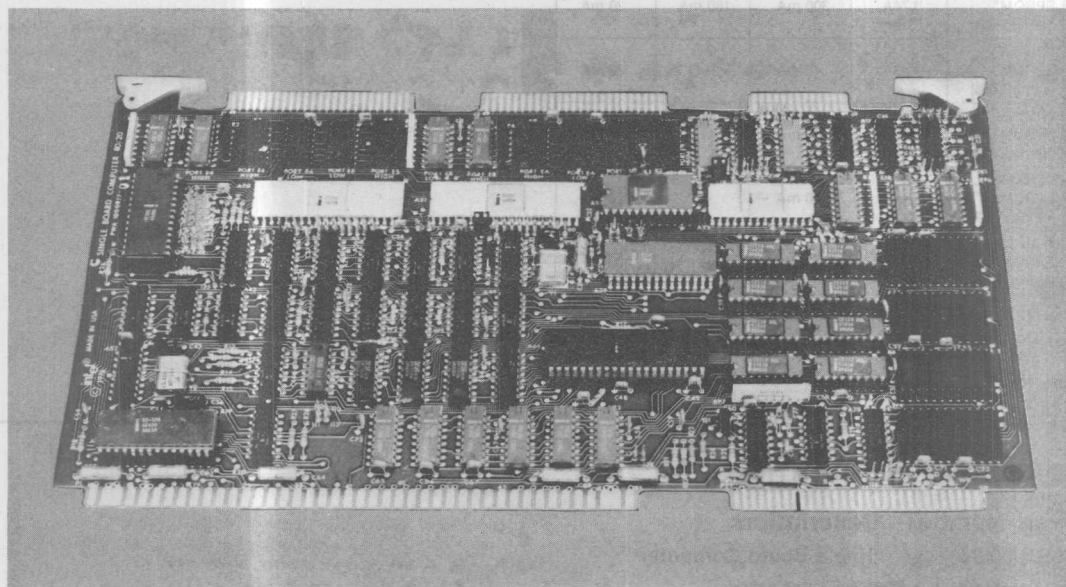
Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.



iSBC 80/20-4 (or pSBC 80/20-4*) SINGLE BOARD COMPUTER

- 8080A CPU used as central processor
- 4K bytes of static read/write memory
- Sockets for up to 8K bytes of erasable reprogrammable or masked read only memory
- 48 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators
- Programmable synchronous/asynchronous RS232C compatible serial interface with fully software selectable baud rate generation
- Full MULTIBUS control logic allowing up to 16 masters to share system bus
- Two programmable 16-bit BCD and binary timers
- Eight-level programmable interrupt control
- Compatible with optional memory and I/O expansion boards
- Auxiliary power bus, memory protect, and power-fail interrupt control logic provided for battery backup RAM requirements

The iSBC 80/20-4 Single Board Computer is a member of Intel's complete line of OEM computer systems which take full advantage of Intel's LSI technology to provide economical, self-contained computer-based solutions for OEM applications. Each iSBC 80/20-4 is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, read/write memory, nonvolatile read only memory, I/O ports and drivers, serial communications interface, priority interrupt logic, two programmable timers, MULTIBUS control logic, and bus expansion drivers all reside on each board.



*Same product, manufactured by Intel Puerto Rico, Inc.

FUNCTIONAL DESCRIPTION

Intel's powerful 8-bit n-channel MOS 8080A CPU, fabricated on a single LSI chip, is the central processor for the iSBC 80/20-4. The 8080A contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators. Minimum instruction execution time is 1.86 microseconds. A block diagram of iSBC 80/20-4 functional components is shown in Figure 1.

Memory Addressing

The 8080A has a 16-bit program counter which allows direct addressing of up to 65,536 bytes of memory. An external stack, located within any portion of read/write memory, may be used as a last-in/first-out storage area for the contents of the program counter, flags, accumulator, and all of the six general purpose registers. A 16-bit stack pointer controls the addressing of this external stack. This stack provides subroutine nesting bounded only by memory size.

Memory Capacity

The iSBC 80/20-4 contains 4K bytes of static read/write memory using Intel low power static RAMs. All on-board RAM read and write operations are performed at maximum processor speed. Power for on-board RAM memory is provided on an auxiliary power bus, and memory protect logic is included for battery backup RAM requirements. Sockets for up to 8K bytes of nonvolatile read only memory are provided on the board. Read only

memory may be added in 1K-byte increments using Intel 2708 erasable and electrically reprogrammable ROMs (EPROMs), or read only memory may be added in 2K-byte increments using Intel 2716 EPROMs. All on-board ROM read operations are performed at maximum processor speed.

Parallel I/O Interface

The iSBC 80/20-4 contains 48 programmable parallel I/O lines implemented using two Intel 8255 programmable peripheral interfaces. The system software is used to configure the I/O lines in any combination of the unidirectional input/output, and bidirectional ports indicated in Table 1. Therefore, the I/O interface may be customized to meet specified peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 48 programmable I/O lines and signal ground lines are brought out to two 50-pin edge connectors that mate with flat, woven, or round cable.

Serial I/O Interface

A programmable communications interface using Intel's 8251 Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC

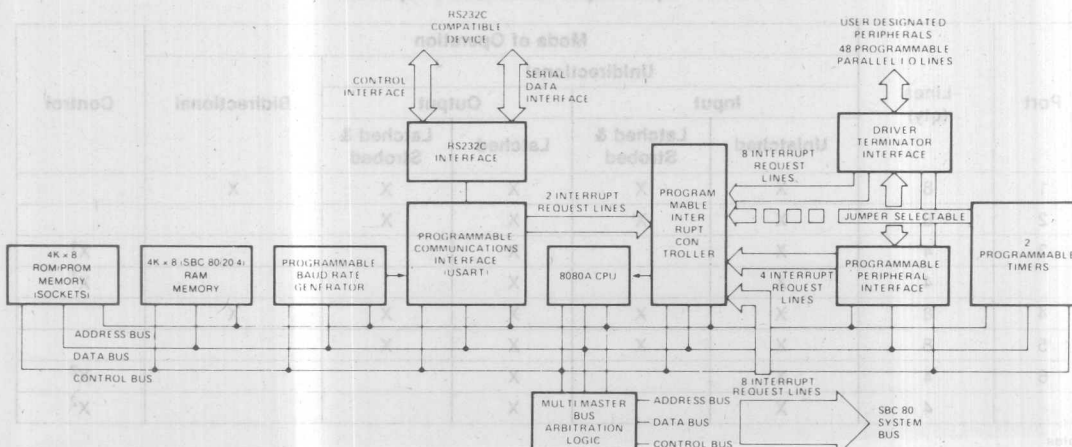


Figure 1. iSBC 80/20 and iSBC 80/20-4 Block Diagram Showing Functional Components

80/20-4 board. A software selectable baud rate generator provides the USART with all common communications frequencies. The USART can be programmed by the system software to select the desired asynchronous or synchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e., synchronous or asynchronous), data format, control character parity, and baud rate are all under program control. The 8251 provides full duplex, double-buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The RS232C compatible interface on each board, in conjunction with the USART, provides a direct interface to RS232C compatible terminals, cassettes, and asynchronous and synchronous modems. The RS232C command lines, serial data lines, and signal ground line are brought out to a 26-pin edge connector that mates with RS232C compatible flat or round cable.

Multimaster Capability

The ISBC 80/20-4 is a full computer on a single board with resources capable of supporting the majority of OEM system requirements. For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically share system tasks with communication over the system bus), the ISBC 80/20-4 provides full MULTI-BUS arbitration control logic. This control logic allows up to three ISBC 80/20-4 or high speed controllers to share the system bus in serial (daisy chain) priority fashion, and up to 16 masters may share the system bus with the addition of an external priority network. Once

bus control is attained, a bus bandwidth of up to 5M bytes/sec may be achieved.

The bus controller provides its own clock which is derived independently from the processor clock. This allows different speed controllers to share resources on the same bus, and transfers via the bus proceed asynchronously. Thus, transfer speed is dependent on transmitting and receiving devices only. This design prevents slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. Once a bus request is granted, single or multiple read/write transfers can proceed at a maximum rate of 5 million data words per second. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed direct-memory-access (DMA) operations and high speed peripheral control, but are by no means limited to these three.

Programmable Timers

The ISBC 80/20-4 board provides three fully programmable and independent BCD and binary 16-bit interval timers/event counters utilizing an Intel 8253 Programmable Interval Timer. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing of these counters is jumper selectable. Each may be independently routed to the programmable interrupt controller, the I/O line drivers and terminators, or outputs from the 8255 programmable peripheral interfaces. The third interval timer in the 8253 provides the programmable baud

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation				Control	
		Unidirectional					Bidirectional
		Input		Output			
		Unlatched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	4	X		X			X ¹
	4	X		X			X ¹
4	8	X	X	X	X	X	
5	8	X	X	X	X		
6	4	X		X			X ²
	4	X		X			X ²

Notes

1. Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.
2. Part of port 6 must be used as a control port when either port 4 or port 5 are used as a latched and strobed input or a latched and strobed output port or port 4 is used as a bidirectional port.

rate generator for the iSBC 80/20-4 RS232C USART serial port. In utilizing the iSBC 80/20-4, the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given time delay or count is needed, software commands to the programmable timers/event counters select the desired function. Seven functions are available, as shown in Table 2. The contents of each counter may be read at any time during system operation with simple read operations for event counting applications, and special commands are included so that the contents of each counter can be used "on the fly".

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low-going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge on counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counting "window" has been enabled or an interrupt may be generated after N events occur in the system.

Interrupt Capability

Operation and Priority Assignments — An Intel 8259 Programmable Interrupt Controller (PIC) provides vectoring for eight interrupt levels. As shown in Table 3, a selection of four priority processing modes is available to the systems designer so that the manner in which requests are processed may be configured to match system requirements. Operating mode and priority

assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from the programmable parallel and serial I/O interfaces, the programmable timers, the system bus, or directly from peripheral equipment. The PIC then determines which of the incoming requests is of the highest priority, determines whether this request is of higher priority than the level currently being serviced, and if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked through storage via software, of a single byte to the interrupt register of the PIC.

Table 3. Programmable Interrupt Modes

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until the next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

Interrupt Addressing — The PIC generates a unique memory address for each interrupt level. These addresses are equally spaced at intervals of 4 or 8 (software selectable) bytes. This 32- or 64-byte block may be located to begin at any 32- or 64-byte boundary in the 65,536-byte memory space. A single 8080 jump instruction at each of these addresses then provides linkage to locate each interrupt service routine independently anywhere in memory.

Interrupt Request Generation — Interrupt requests may originate from 26 sources. Four jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface when a byte of information is ready to be transferred to the CPU (i.e., input buffer is full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). Two jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transfer to the CPU (i.e., receive channel buffer is full), or a character is ready to be transmitted (i.e., transmit channel data buffer is empty). A jumper selectable request can be generated by each of the programmable timers. Nine additional interrupt request lines are available to the user for direct interface to user designated peripheral devices via the system bus, and eight interrupt request lines may be jumper routed directly from peripherals via the parallel I/O driver/terminator section.

Power-Fail Control — Control logic is also included for generation of a power-fail interrupt which works in conjunction with the AC-low signal from iSBC 635 Power Supply or equivalent.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. High speed integer and floating-point arithmetic capabilities may be added by using the iSBC 310A High Speed Mathematics Unit. Memory may be expanded to 65,536 bytes by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers as subsystems. Modular expandable backplanes and cardcages are available to support multiboard systems.

Real-Time Software

The iSBC 80/20-4 is totally compatible with Intel's iRMX 80 Real-Time Multi-Tasking Executive. iSBC 80/20-4 based user programs (tasks) can take advantage of the iRMX 80 executive to do all necessary scheduling, inter-task communication, and memory space allocation. iRMX 80 also provides standard I/O support software such as disk file handling, Intel analog board handling, and terminal handling.

System Development Capability

The development cycle of iSBC 80/20-4-based products may be significantly reduced using Intel's system development tools available today. For those not requiring hardware emulation capability, Intel provides a new low cost microcomputer development system. The iPDS, Personal Development System, provides low cost system development for the iSBC 80/20-4 board, while at the same time providing personal computer capability for the engineer. The Intellec Series II family of compatible microcomputer development systems provides a range of capability from a low cost disk-based edit debug workstation to a high performance, fully compatible hard-

disk-based software development system. A unique in-circuit emulator (ICE-80) option provides the capability of developing and debugging software directly on the iSBC 80/20-4 board.

Programming Capability

PL/M-80 — Intel's high level programming language, PL/M, is also available as a resident Intellec microcomputer development system option. PL/M provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M programs can be written in a much shorter time than assembly language programs for a given application.

FORTTRAN-80 — For applications requiring computational and formatted I/O capabilities, the high level FORTTRAN-80 programming language is also available as a resident option of the Intellec system. The FORTTRAN compiler produces relocatable object code that may be easily linked with PL/M or assembly language program modules. This gives the user a wide flexibility in developing software.

BASIC-80 — A high level language interpreter with extended disk capabilities which operates under the RMX/80 Real-Time Multi-Tasking Executive and translates BASIC-80 source programs into an internally executable form. This language interpreter, provided as a set of linkable object modules, is ideally suited to the OEM who requires a pass through programming language. The BASIC-80 programs may be created, stored and interpreted on the iSBC 80 based system. The BASIC-80 language has a rich complement of statements, functions, and commands to program applications requiring a full range of 1) string manipulation and disk I/O for data processing, 2) single and double precision floating point and array handling for numeric analysis, or 3) port I/O with mask operations controlled through bit-wise Boolean logical operators.

SPECIFICATIONS

Word Size

Instruction — 8, 16, or 24 bits

Data — 8 bits

Cycle Time

Basic Instruction Cycle — 1.86 μ s

Note

Basic instruction cycle is defined as the fastest instruction (i.e., four clock cycles).

Memory Addressing

On-Board ROM/EPROM — 0-0FFF (2708) or 0-1FFF (2716)

On-Board RAM — 4K bytes ending on a 16K boundary (e.g., 3FFF_H, 7FFF_H, BFFF_H, ... FFFF_H)

Memory Capacity

On-Board ROM/EPROM — 8K bytes (sockets only)

On-Board RAM — 4K bytes

Off-Board Expansion — Up to 65,536 bytes in user specified RAM, ROM, and EPROM

Note

ROM/EPROM may be added in 1K or 2K-byte increments.

I/O Addressing

On-Board Programmable I/O (see Table 1)

Port	8255 No. 1			8255 No. 2			8255 No. 1 Control	8255 No. 2 Control	USART Data	USART Control
	1	2	3	4	5	6				
Address	E4	E5	E6	E8	E9	EA	E7	EB	EC	ED

I/O Capacity

Parallel — 48 programmable lines (see Table 1)

Note

Expansion to 504 input and 504 output lines can be accomplished using optional I/O boards.

Serial Communications Characteristics

Synchronous — 5–8 bit characters; internal or external character synchronization; automatic sync insertion

Asynchronous — 5–8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection

Baud Rates

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)	
	Synchronous	Asynchronous
153.6	—	± 16 ± 64
76.8	—	9600 2400
38.4	38400	4800 1200
19.2	19200	2400 600
9.6	9600	1200 300
4.8	4800	600 150
2.4	2400	300 75
1.76	1760	150 —
		110 —

Note

Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register.

Register Address (hex notation, I/O address space)

DE Baud rate register

Note

Baud rate factor (16 bits) is loaded as two sequential output operations to same address (DEH).

Interrupts

Register Addresses (hex notation, I/O address space)

DA Interrupt request register

DA In-service register

DB Mask register

DA Command register

DB Block address register

DA Status (polling register)

Note

Several registers have the same physical address; sequence of access and one data bit of control word determine which register will respond.

Timers

Register Addresses (hex notation, I/O address space)

DF Control register

DC Timer 1

DD Timer 2

Note

Timer counts loaded as two sequential output operations to same address, as given.

Input Frequencies

Reference	Event Rate
1.0752 MHz ± 10% (0.930 μs period, nominal)	1.1 MHz max

Note

Maximum rate for external events in event counter function.

Output Frequencies/Timing Intervals

Function	Single Timer/Counter		Dual Timer/Counter (Two Timers Cascaded)	
	Min	Max	Min	Max
Real-time interrupt	1.86 μs	60.948 ms	3.72 μs	1.109 hr
Programmable one-shot	1.86 μs	60.948 ms	3.72 μs	1.109 hr
Rate generator	16.407 Hz	537.61 kHz	0.00025 Hz	268.81 kHz
Square-wave rate generator	16.407 Hz	537.61 kHz	0.00025 Hz	268.31 kHz
Software triggered strobe	1.86 μs	60.948 ms	3.72 μs	1.109 hr
Hardware triggered strobe	1.86 μs	60.948 ms	3.72 μs	1.109 hr

Interfaces

Bus — All signals TTL compatible

Parallel I/O — All signals TTL compatible

Interrupt Requests — All TTL compatible

Timer — All signals TTL compatible

Serial I/O — RS232C compatible, data set configuration

System Clock (8080A CPU)

2.1504 MHz ± 0.1%

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM for systems requiring battery backup of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power-down sequences.

Connectors

Interface	Double-Sided Pins (qty)	Centers (in.)	Mating Connectors*
MULTIBUS System Bus	86	0.156	ELFAB BS1562043PBB Viking 2KH43/9AMK12 Soldered PCB Mount EDAC 337086540201 ELFAB BW1562D43PBB EDAC 337086540202 ELFAB BW1562A43PBB Wire Wrap
Auxiliary Bus	60	0.100	EDAC 345060524802 ELFAB BS1020A30PBB EDAC 345060540201 ELFAB BW1020D30PBB Wire Wrap
Parallel I/O (2)	50	0.100	3M 3415-001 Flat Crimp GTE Sylvania 6AD01251A1DD Soldered
Serial I/O	26	0.100	AMP 15837151 EDAC 345026520202 PCB Soldered 3M 3462-0001 AMP 88373-5 Flat Crimp

*Note: Connectors compatible with those listed may also be used.

I/O Drivers — The following line drivers are all compatible with the I/O driver sockets on the iSBC 80/20-4.

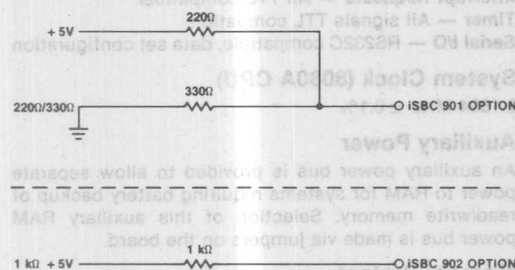
Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

Note

I = inverting; NI = non-inverting; OC = open collector.

Ports 1 and 4 have 20 mA totem-pole bidirectional drivers and 1 k Ω terminators.

I/O Terminators — 220Ω/330Ω divider or 1 kΩ pull-up



Bus Drivers

Driver	Characteristic	Sink Current (mA)
Data	Tri-state	50
Address	Tri-state	50
Commands	Tri-state	32

ORDERING INFORMATION

Part Number	Description
-------------	-------------

SBC 80/20-4 Single Board Computer with 4K bytes RAM

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.50 in. (1.26 cm)

Weight — 14 oz (397.6 gm)

Electrical Characteristics

DC Power Requirements

Voltage ($\pm 5\%$)	Without PROM ¹ (max)	With 4K PROM ² (max)	With ISBC 530 ³ (max)	RAM Only ⁴ (max)	With 8K PROM ⁵ (max)
$V_{CC} = +5V$	$I_{CC} = 4.0A$	4.9A	4.9A	1.1A	5.2A
$V_{DD} = +12V$	$I_{DD} = 90mA$	350mA	450mA	—	90mA
$V_{BB} = -5V$	$I_{BB} = 2mA$	180mA	180mA	—	2mA
$V_{AA} = -12V$	$I_{AA} = 20mA$	20mA	120mA	—	20mA

Notes

1. Does not include power required for optional PROM, I/O drivers, and I/O terminators.

2. With four 2708 EPROMs and 220 Ω /330 Ω input terminators installed for 32 I/O lines, all terminator inputs low.

3. With four 2708 EPROMs, 220 Ω /330 Ω input terminators installed for 32 I/O lines, all terminator inputs low, and iSBC 530 Teletypewriter Adapter drawing power from serial port connector.

4. RAM chips powered via auxiliary power bus.

5. With four 8716 EPROMs and eight 220 Ω /330 Ω input terminators installed, all terminator inputs low.

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Reference Manual

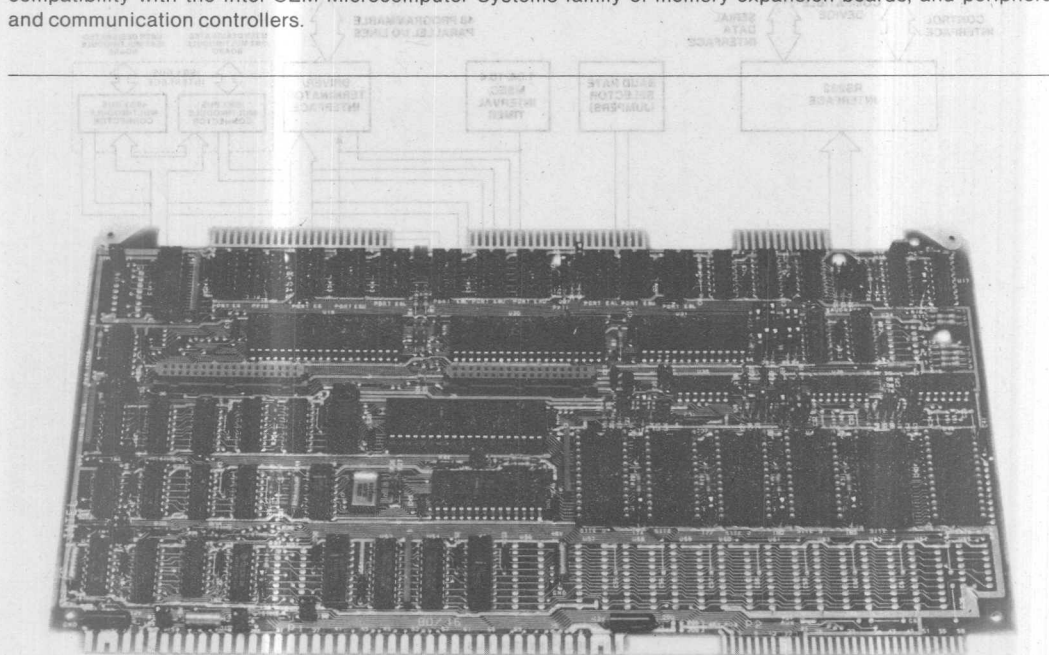
9800317D — iSBC 80/20-5 Hardware Reference Manual
(NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor, office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

iSBC® 80/16 SINGLE BOARD COMPUTER

- 8080A CPU
- Two iSBX™ bus connectors for iSBX™ MULTIMODULE™ board expansion
- Six 28-pin JEDEC compatible sockets for RAM/EPROM/E²PROM hold up to 64KB of memory
- 2K x 8 static RAM in one of the six JEDEC sockets
- 48 programmable parallel I/O lines
- Programmable synchronous/asynchronous communications interface with RS232C compatibility
- Single level interrupt with 12 interrupt sources
- 1.04 or 10.4 millisecond timer
- Limited master MULTIBUS® interface

The Intel® iSBC® 80/16 board is a member of Intel's complete line of OEM microcomputer systems which take full advantage of Intel's LSI technology to provide economical, self-contained computer-based solutions for OEM applications. The iSBC 80/16 board is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, iSBX bus interface, read/write memory, memory sockets, I/O ports and drivers, and serial communications interface all reside on the board. MULTIBUS control logic is included to offer compatibility with the Intel OEM Microcomputer Systems family of memory expansion boards, and peripheral and communication controllers.



The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products: Intel, Inteltec, ICE, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTIMODULE, Multichannel, iCS, and iPDS. Intel Corporation Assumes No Responsibility for the use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Patent Licenses are Implied.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Intel's 8-bit n-channel MOS 8080A CPU, fabricated on a single LSI chip, is the central processor for the iSBC 80/16 board. The 8080A contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators. A block diagram of iSBC 80/16 board functional components is shown in Figure 1.

iSBX™ MULTIMODULE™ On-Board Expansion

Two 8-bit iSBX MULTIMODULE connectors are provided on the iSBC 80/16 microcomputer. Through these connectors, additional on-board I/O functions

may be added. iSBX MULTIMODULES optimally support functions provided by VLSI peripheral components such as additional parallel and serial I/O, analog I/O, small mass storage device controllers (e.g., cassettes and floppy disks), and other custom interfaces to meet specific needs. By mounting directly on the single board computer, less interface logic, less power, simpler packaging, higher performance, and lower cost result when compared to other alternatives such as MULTIBUS form factor compatible boards. The iSBX connectors on the iSBC 80/16 board provide all signals necessary to interface to the local on-board bus. A broad range of iSBX MULTIMODULE options are available in this family from Intel. Custom iSBX modules may also be designed for use on the iSBC 80/16 board. An iSBX bus interface specification is available from Intel.

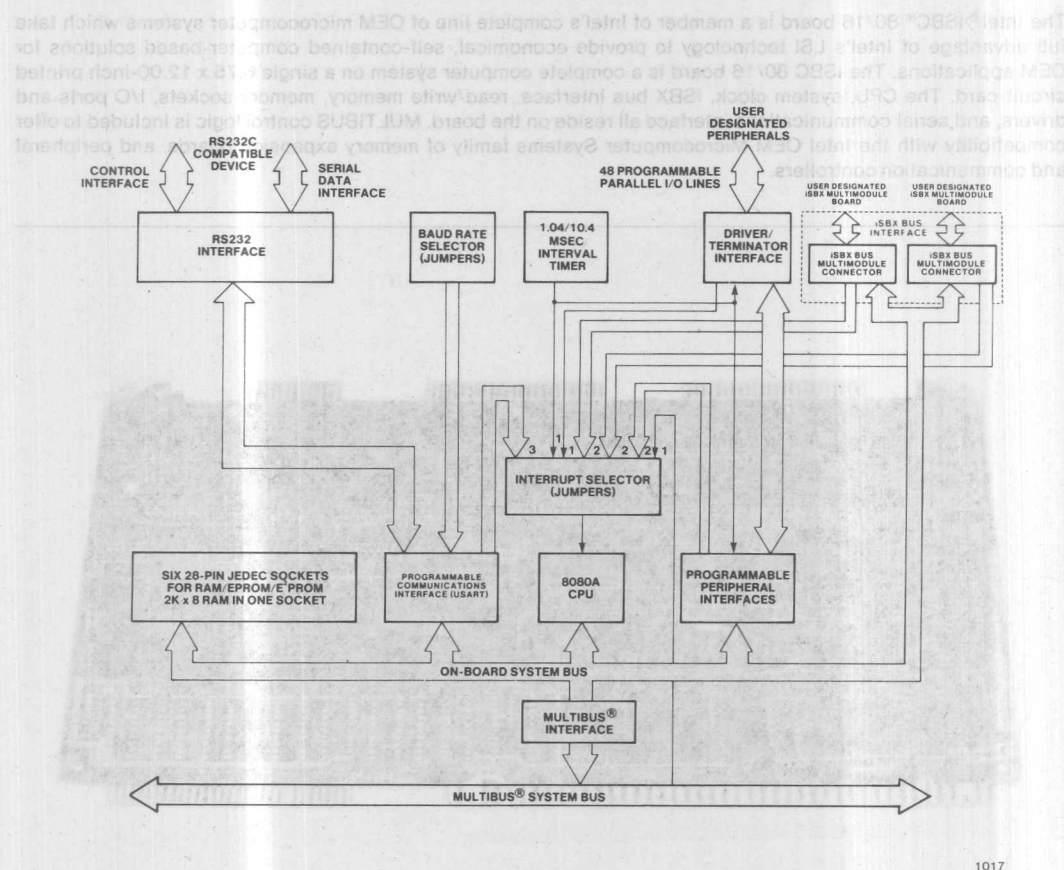


Figure 1. iSBC® 80/16 Single Board Computer Block Diagram

Memory Addressing

The 8080A has a 16-bit program counter which allows direct addressing of up to 64K bytes of memory. An external stack, located within any portion of read/write memory, may be used as a last-in/first-out storage area for the contents of the program counter, flags, accumulator, and all of the six general purpose registers. A 16-bit stack pointer controls the addressing of this external stack. This stack provides subroutine nesting bounded only by memory size.

Memory Capacity

The iSBC 80/16 board contains six 28-pin compatible byte wide memory sockets. These are compatible with 2K x 8 (Intel 2716), 4K x 8 (Intel 2732), 8K x 8 (Intel 2764), and 16K x 8 (Intel 27128) EPROMs, 2K x 8 and 8K x 8 static RAMs, and 2K x 8 (Intel 2817, 2817A) E²PROMs. Modification of the address decode PROM allows compatibility with 1K x 8 (Intel 2708) EPROMs and future 32K x 8 EPROMs. The sockets are configured in three sets of two. Each pair may be configured for similar devices which may be EPROM, RAM or E²PROM devices. Using 8K x 8 static RAMs, a maximum of 32K bytes of RAM can be installed on the iSBC 80/16 board. Using 16 x 8 EPROMs (27128s), a maximum of 64K bytes of EPROM can be installed on-board. In this instance the on-board RAM would then overlay the EPROM since the maximum address space of the iSBC

80/16 board is 64K bytes. All on-board RAM and EPROM operations are performed at maximum processor speed.

Parallel I/O Interface

The iSBC 80/16 board contains 48 programmable parallel I/O lines using two Intel 8255A programmable peripheral interfaces. The system software is used to configure the I/O lines in any combination of unidirectional input/output, and bi-directional ports as indicated in Table 1. Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further enhanced by selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 48 programmable I/O lines and signal ground lines are brought out to two 50-pin edge connectors that mate with flat cable or round cable.

Serial I/O Interface

A programmable communications interface using the Intel® 8251A Universal Synchronous/ Asynchronous Receiver/Transmitter (USART) is contained on the board. A jumper selectable baud rate generator

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation					
		Unidirectional				Bidirectional	Control
		Input		Output			
		Unlatched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	8	X		X			X ¹
4	8	X		X			
5	8	X		X			
6	4	X		X			
	4	X		X			

Notes

Port 3 must be used as a control port when either port 1 or port 2 are used as latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

provides the USART with most common communications frequencies. The USART can be programmed by the system software to select the desired synchronous or asynchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e. synchronous or asynchronous), data format, control character format and parity are all under program control. The 8251A provides full duplex, double-buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The compatible interface provides a direct interface to CRTs, RS232C compatible cassettes, and asynchronous and synchronous modems. The RS232C control lines, serial data lines, and signal ground lines are brought out to a 26-pin edge connector that mates with RS232C compatible flat or round cable.

Interrupt Capability

Interrupt requests may originate from 12 sources. Two jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface when a byte of information is ready to be transferred to the CPU (i.e., input buffer is full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). Three jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU (i.e., receive channel buffer is full), a character is ready to be transmitted (i.e., the USART is ready to accept a character from the CPU), or when the transmitter is empty (i.e., the USART has no character to transmit). Two interrupt request lines may be interfaced directly to user designated peripheral devices; one via the MULTIBUS system bus and the other via the I/O edge connector. One jumper selectable interrupt request may originate from the interval timer. Two general purpose interrupt requests are jumper selectable from each iSBX interface. These two signals permit a user installed iSBX board to interrupt the 8080A CPU. The 12 interrupt request lines share a single CPU interrupt level. When an interrupt request is recognized, a restart instruction (RST 7) is generated. The processor responds by suspending program execution and executing a user defined interrupt service routine originating at location 38H.

Interval Timer

A 1.04 or 10.4 millisecond timer is available for interval interrupts or as a clock output to the parallel I/O connector and is jumper selectable. The timer output is jumper selectable to the programmable parallel interface, the parallel I/O connector (J1), or directly to the 8080A CPU.

MULTIBUS® System Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS system compatible expansion boards. Memory may be expanded to 65,536 bytes by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. In addition, the iSBC 80/16 board performs as a limited bus master in that it must occupy the lowest priority when used with other MULTIBUS masters. The bus master may take control of the MULTIBUS system bus by halting the iSBC 80/16 board program execution. Mass storage capability may be achieved by adding single density diskette, double density diskette, or hard disk controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

Real-Time Software

The iRMX 80 executive, which contains all major real-time facilities including priority-based system resource allocation, intertask communication and control, interrupt driven control for standard I/O devices, and interrupt handling, occupies 2K bytes of memory which can be stored on-board in EPROM. Optional linkable and relocatable modules for console control (CRT or TTY), disk file system, and analog subsystems are provided with the iRMX 80 package. User configurability is aided on the Intellec microcomputer development system by the Interactive Configuration Utility program provided with the iRMX 80 package.

System Development Capability

The development cycle of iSBC 80/16-based products may be significantly reduced using Intel's system development tools available today. For those not requiring hardware emulation capability, Intel provides a new low cost microcomputer development system. The iPDS, Personal Development System, provides low cost system development for the iSBC 80/16 board, while at the same time providing personal computer capability for the engineer. The Intellec Series II family of compatible microcomputer development systems provides a range of capability from a low cost disk-based edit debug workstation to a high performance, fully compatible hard-disk-based software development system. A unique in-circuit emulator (ICE-80) option provides the capability of developing and debugging software directly on the iSBC 80/16 board.

Programming Capability

PL/M-80 — Intel's high level programming language, PL/M, is available as a resident compiler on Intel's line of development systems. PL/M provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M programs can be written in a much shorter time than assembly language programs for a given application.

FORTRAN 80 — For applications requiring computational and formatted I/O capabilities, the ANSI 77 standard high level FORTRAN 80 programming language is available as a resident compiler on Intel's line of development systems. The FORTRAN compiler produces relocatable object code that may be easily linked with PL/M or assembly language program modules. In addition, the iSBC 801 FORTRAN 80 run-time package is a complete, ready-to-use set of linkable object modules which are fully compatible with iRMX 80 systems. The modules, when combined with the

FORTRAN 80 coded application, provide the appropriate interfaces to the disk file and terminal I/O of iRMX 80, and to the iSBX 332 math module for applications requiring high speed math.

BASIC-80 — A high level language interpreter is available with extended disk capabilities which operates under the iRMX 80 Real-Time Multitasking Executive and translates BASIC-80 source programs into an internally executable form. This language interpreter, provided as a set of linkable object modules, is ideally suited to the OEM who requires a pass through programming language. The BASIC-80 programs may be created, stored, and interpreted on the iSBC 80 based systems using the iSBC 802 BASIC-80 Configurable iRMX 80 Disk-Based Interpreter. The iSBC 802 Interpreter has a complete ready-to-use set of linkable object modules which are fully compatible with Intel's iRMX 80 Real-Time Multitasking Executive Software. The modules provide interfaces to disk file and terminal I/O, software floating point, or interface to other routines provided by the user.

SPECIFICATIONS

Word Size

Instruction — 8, 16, or 24 bits
Data — 8 bits

Cycle Time

Basic Instruction Cycle — 1.95 usec, 2.048 MHz

Note: Basic instruction cycle is defined as the fastest instruction (i.e., four clock cycles).

Memory Addressing

On-board EPROM

- 0-1FFF using 2716* (8KB)
- 0-3FFF using 2732 (16KB)
- 0-7FFF using 2764 (32KB)
- 0-bottom of RAM using 27128 (64KB minus on-board RAM)

On-board RAM

- 3000-3FFF (4KB), 3800-3FFF installed* (2KB) with 2716
- 4000-4FFF (4KB) with 2732
- X000-FFFF (4KB, 16KB or 32KB) with 2764 or 27128

*as shipped from factory

Memory Capacity

On-Board EPROM

64K bytes (sockets only)

May be added in 2K (using Intel 2716), 4K (using Intel 2732), 8K (using Intel 2764), or 16K (using Intel 27128) byte increments. May be added in 1K byte (using Intel 2708) increments if decode prom is reprogrammed.

On-Board RAM

2K bytes in one of the six JEDEC sockets. Expansion to a maximum of 32K bytes using 8Kx8 static RAMs in four sockets.

Off-Board Expansion

Up to 64K bytes using user specified combinations of RAM, ROM, EPROM, and E²PROM.

I/O Addressing

On-board Programmable I/O

Device	I/O Address
8255A No. 1	
Port A	E4
Port B	E5
Port C	E6
Control	E7
8255A No. 2	
Port A	E8
Port B	E9
Port C	EA
Control	EB
8251A	
Data	EC
Control	ED
ISBX™ Multimodule™ J5	
MCS0	F0-F7
MCS1	F8-FF
ISBX™ Multimodule™ J4	
MCS0	CO-C7
MCS1	C8-CF

I/O Capacity

Parallel - 48 programmable lines

Serial - 1 transmit, 1 receive

MULTIMODULE - 2 iSBX MULTIMODULE Boards

Serial Baud Rates

Frequency (kHz) (Jumper Selectable)	Baud Rate (Hz)	
	Synchronous	Asynchronous (Program Selectable)
307.2	—	16 64
153.6	—	19200 4800
76.8	—	9600 2400
38.4	—	4800 1200
19.2	38400	2400 600
9.6	19200	1200 300
4.8	9600	600 150
	4800	300 75
460.8	—	— 7200
230.4	—	14400 3600
115.2	—	7200 1800
57.6	—	3600 900
28.8	28800	1800 450
14.4	14400	900 225
7.2	7200	450 112.5

Serial Communications Characteristics

Synchronous - 5-8 bit characters; internal or external character synchronization; automatic sync insertion.

Asynchronous - 5-8 bit characters; break character generation; 1, 1-1/2, or 2 stop bits; false start bit detectors.

Interrupts

Single-level with on-board logic that automatically vectors the processor to location 38H using a restart instruction (RST, 7). Interrupt requests may originate from user specified I/O (2); the programmable peripheral interface (2); two from each iSBX MULTIMODULE board (4); the programmable communications interface (3); or the interval timer (1).

Electrical Characteristics

Voltage	Without EPROM
$V_{CC} = +5V \pm 5\%$	$I_{CC} = 1.95A$
$V_{DD} = +12V \pm 5\%$	$I_{DD} = 160 mA$
$V_{BB} = -5V \pm 5\%$	$I_{BB} = 0$
$V_{AA} = -12V \pm 5\%$	$I_{AA} = 100 mA$

Notes:

- Does not include power required for byte-wide memory devices, I/O drivers, or I/O terminators.
- Max I_{CC} for memory devices is 420 mA
- V_{BB} only required for 2708s.

Interfaces

MULTIBUS - All signals TTL compatible

iSBX Bus - All signals TTL compatible

Parallel I/O - All signals TTL compatible

Serial I/O - RS232C interface

Interrupt Requests - All TTL compatible (active-low)

Clocks

System Clock - 2.048 MHz $\pm 0.1\%$

Interval Timer - 1.042 msec $\pm 0.1\%$

or 10.42 msec $\pm 0.1\%$

with interrupt latch

Connectors

Interface	Double-Sided Pins (qty.)	Centers (in.)	Mating Connectors*
MULTIBUS® System Bus	86	0.156	ELFAB BS1562043PBB Viking 2KH43/9AMK12 Soldered PCB Mount EDAC 337086540201 ELFAB BW1562D43PBB EDAC 337086540202 ELFAB BW1562A43PBB Wire Wrap
Auxiliary Bus	60	0.100	EDAC 345060524802 ELFAB BS1020A30PBB EDAC 345060540201 ELFAB BW1020D30PBB Wire Wrap
iSBX™ Bus (2)	36	0.100	Viking VSBX 000292-0001
Parallel I/O (2)	50	0.100	3M3415-001 Flat Crimp GTE Sylvania 6AD01251A1DD Soldered
Serial I/O	26	0.100	AMP 15837151 EDAC 345026520202 PCB Soldered 3M 3462-0001 AMP 88373-5 Flat Crimp

*Note: Connectors compatible with those listed may also be used.

Physical Characteristics

Width - 12.00 in. (30.48 cm)
Height - 6.75 in. (17.15 cm)
Depth - 0.5 in. (1.27 cm)
Weight - 13 oz. (371 gm)

Line Drivers And Terminators

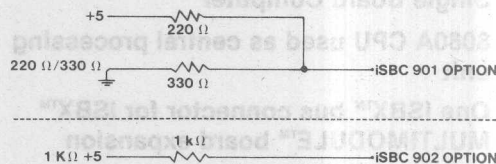
I/O Drivers - The following line drivers and terminators are all compatible with the I/O driver sockets on the iSBC 80/16 board:

Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

Note: I - inverting, NI - non-inverting, OC - open collector.

Port 1 has 25 mA totem pole drivers and 1 k Ω terminators.

I/O Terminators - 220 Ω /330 Ω divider or 1 k Ω pull up.



MULTIBUS® Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-State	32
Address	Tri-State	24
Commands	Tri-State	32

Environmental Characteristics

Operating Temperature - 0°C to 55°C
Relative Humidity - to 90%, non-condensing

Equipment Supplied

iSBC 80/16 Single Board Computer
iSBC 80/16 Schematic

Reference Manual

9803119-01 - iSBC 80/16 Single Board Computer
Hardware Reference Manual
(NOT SUPPLIED).

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California. 95051.

ORDERING INFORMATION

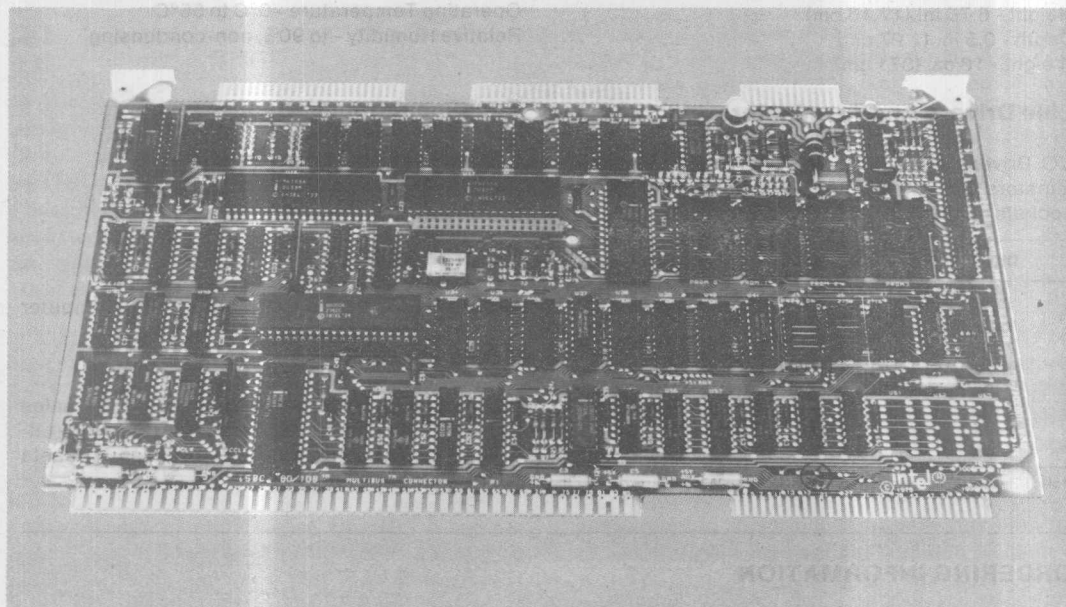
Part Number	Description
SBC 80/16	Single Board Computer



iSBC™ 80/10B (or pSBC 80/10B*) SINGLE BOARD COMPUTER

- Upward compatible with iSBC™ 80/10A Single Board Computer
- 8080A CPU used as central processing unit
- One iSBX™ bus connector for iSBX™ MULTIMODULE™ board expansion
- 1K byte of read/write memory with sockets for expansion up to 4K bytes
- Sockets for up to 16K bytes of read only memory
- 48 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators
- Programmable synchronous/asynchronous communications interface with selectable RS232C or teletypewriter compatibility
- Single level interrupt with 11 interrupt sources
- Auxiliary power bus and power-fail interrupt control logic for RAM battery backup
- 1.04 millisecond interval timer
- Limited master MULTIBUS® interface

The Intel® iSBC 80/10B board is a member of Intel's complete line of OEM microcomputer systems which take full advantage of Intel's LSI technology to provide economical, self-contained computer-based solutions for OEM applications. The iSBC 80/10B board is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, iSBX bus interface, read/write memory, read only memory sockets, I/O ports and drivers, serial communications interface, bus control logic, and drivers all reside on the board.



*Same product, manufactured by Intel Puerto Rico, Inc.

FUNCTIONAL DESCRIPTION

Intel's powerful 8-bit n-channel MOS 8080A CPU, fabricated on a single LSI chip, is the central processor for the iSBC 80/10B board. The 8080A contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators. A block diagram of iSBC 80/10B board functional components is shown in Figure 1.

ISBX Bus MULTIMODULE Board Expansion

The new iSBX bus interface brings an entirely new dimension to system design offering incremental

on-board expansion with small iSBX boards. One iSBX bus connector interface is provided to accomplish plug-in expansion with any iSBX MULTIMODULE board. iSBX boards are available to provide expansion equivalent to the I/O available on the iSBC 80/10B board or the user may configure entirely new functionality such as math directly on-board. The iSBX 350 programmable I/O MULTIMODULE board provides 24 I/O lines using an 8255A programmable peripheral interface. Therefore, the iSBX 350 module together with the iSBC 80/10B board may offer 72 lines of programmable I/O. Alternately, a serial port may be added using the iSBX 351 serial I/O multimodule board or math may be configured on-board with the iSBX 332 floating point math MULTIMODULE board.

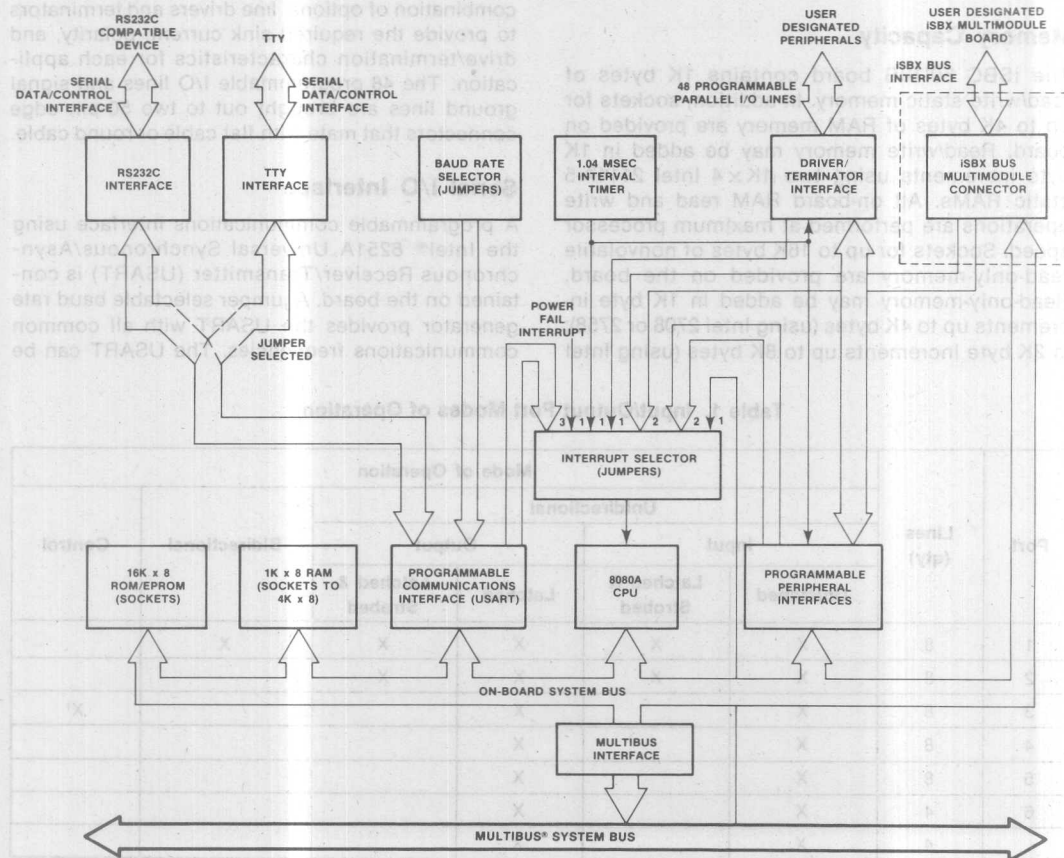


Figure 1. iSBC™ 80/10B Single Board Computer Block Diagram

The iSBX board is a logical extension of the on-board programmable I/O and is accessed by the iSBC 80/10B single board computer as common I/O port locations. The iSBX board is coupled directly to the 8080A CPU and therefore becomes an integral element of the iSBC 80/10B single board computer providing optimum performance.

Memory Addressing

The 8080A has a 16-bit program counter which allows direct addressing of up to 64K bytes of memory. An external stack, located within any portion of read/write memory, may be used as a last-in/first-out storage area for the contents of the program counter, flags, accumulator, and all of the six general purpose registers. A 16-bit stack pointer controls the addressing of this external stack. This stack provides subroutine nesting bounded only by memory size.

Memory Capacity

The iSBC 80/10B board contains 1K bytes of read/write static memory. In addition, sockets for up to 4K bytes of RAM memory are provided on board. Read/write memory may be added in 1K byte increments using two 1K \times 4 Intel 2114A-5 static RAMs. All on-board RAM read and write operations are performed at maximum processor speed. Sockets for up to 16K bytes of nonvolatile read-only-memory are provided on the board. Read-only-memory may be added in 1K byte increments up to 4K bytes (using Intel 2708 or 2758); in 2K byte increments up to 8K bytes (using Intel

2716); or in 4K byte increments up to 16K bytes (using Intel 2732). All on-board ROM or EPROM read operations are performed at maximum processor speed.

Parallel I/O Interface

The iSBC 80/10B board contains 48 programmable parallel I/O lines implemented using two Intel 8255A programmable peripheral interfaces. The system software is used to configure the I/O lines in any combination of unidirectional input/output, and bidirectional ports indicated in Table 1. Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 48 programmable I/O lines and signal ground lines are brought out to two 50-pin edge connectors that mate with flat cable or round cable.

Serial I/O Interface

A programmable communications interface using the Intel® 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the board. A jumper selectable baud rate generator provides the USART with all common communications frequencies. The USART can be

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation					
		Unidirectional				Bidirectional	Control
		Input		Output			
		Unlatched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	8	X		X			X ¹
4	8	X		X			
5	8	X		X			
6	4	X		X			
	4	X		X			

Notes

Port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

programmed by the system software to select the desired synchronous or asynchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format and parity are all under program control. The 8251A provides full duplex, double-buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The inclusion of jumper selectable TTY or RS232C compatible interfaces on the board, in conjunction with the USART, provides a direct interface to teletypes, CRTs, RS232C compatible cassettes, and asynchronous and synchronous modems. The RS232C or TTY command lines, serial data lines, and signal ground lines are brought out to a 26-pin edge connector that mates with RS232C compatible flat or round cable.

Interrupt Capability

Interrupt requests may originate from 11 sources. Two jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface when a byte of information is ready to be transferred to the CPU (i.e., input buffer is full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). Three jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU (i.e., receive channel buffer is full), a character is ready to be transmitted (i.e., the USART is ready to accept a character from the CPU), or when the transmitter is empty (i.e., the USART has no character to transmit). These five interrupt request lines are all maskable under program control. Two interrupt request lines may be interfaced directly to user designated peripheral devices; one via the MULTIBUS system bus and the other via the I/O edge connector. One jumper selectable interrupt request may be interfaced to the power-fail interrupt control logic. One jumper selectable interrupt request may originate from the interval timer. Two general purpose interrupt requests are jumper selectable from the ISBX interface. These two signals permit a user installed MULTIMODULE board to interrupt the 8080A CPU. The eleven interrupt request lines share a single CPU interrupt level. When an interrupt request is recognized, a restart instruction (RESTART 7) is generated. The processor responds by suspending program execution and executing a user defined interrupt service routine originating at location 38₁₆.

Power-Fail Control

A power-fail interrupt may be detected through the AC-low signal generated by the power supply. This

signal may be configured to interrupt the 8080A CPU to initiate an orderly power down instruction sequence.

Interval Timer

A 1.04 millisecond timer is available for interval interrupts or as a clock output to the parallel I/O connector. The timer output is jumper selectable to the programmable parallel interface, the parallel I/O connector (J1), or directly to the 8080A CPU.

MULTIBUS® System Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS™ system compatible expansion boards. Memory may be expanded to 65,536 bytes by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. In addition, the ISBC 80/10B board performs as a limited bus master in that it must occupy the lowest priority when used with other MULTIBUS masters. The bus master may take control of the MULTIBUS system bus by halting the ISBC 80/10B board program execution. Mass storage capability may be achieved by adding single density diskette, double density diskette, or hard disk controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

Real-Time Software

The iRMX 80 executive, which contains all major real-time facilities including priority-based system resource allocation, intertask communication and control, interrupt driven control for standard I/O devices, and interrupt handling, occupies 2K bytes of memory which can be stored on-board in EPROM. Optional linkable and relocatable modules for console control (CRT or TTY), disk file system, and analog subsystems are provided with the iRMX 80 package. User configurability is aided on the Intellec microcomputer development system by the Interactive Configuration Utility program provided with the iRMX 80 package.

System Development Capability

The development cycle of ISBC 80/10B-based products may be significantly reduced using Intel's system development tools available today. For those not requiring hardware emulation capability, Intel provides a new low cost microcomputer development system. The iPDS, Personal Development System, provides low cost system develop-

ment for the iSBC 80/10B board, while at the same time providing personal computer capability for the engineer. The Intellec Series II family of compatible microcomputer development systems provides a range of capability from a low cost disk-based edit debug workstation to a high performance, fully compatible hard-disk-based software development system. A unique in-circuit emulator (ICE-80) option provides the capability of developing and debugging software directly on the iSBC 80/10B.

Programming Capability

PL/M-80 — Intel's high level programming language, PL/M, is also available as a resident Intellec microcomputer development system option. PL/M provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M programs can be written in a much shorter time than assembly language programs for a given application.

FORTRAN-80 — For applications requiring computational and formatted I/O capabilities, the ANSI 77 standard high level FORTRAN-80 programming language is available as a resident option of the Intellec system. The FORTRAN compiler produces relocatable object code that may be easily linked

with PL/M or assembly language program modules. In addition, the iSBC 801 FORTRAN-80 run-time package is a complete, ready-to-use set of linkable object modules which are fully compatible with iRMX 80 systems. The modules, when combined with the FORTRAN-80 coded application, provide the appropriate interfaces to the disk file and terminal I/O of iRMX 80, and to the iSBC 310A Math Unit for applications requiring high speed math.

BASIC-80 — A high level language interpreter is available with extended disk capabilities which operates under the iRMX 80 Real-Time Multitasking Executive and translates BASIC-80 source programs into an internally executable form. This language interpreter, provided as a set of linkable object modules, is ideally suited to the OEM who requires a pass through programming language. The BASIC-80 programs may be created, stored, and interpreted on the iSBC 80 based systems using the iSBC 802 BASIC-80 Configurable iRMX 80 Disk-Based Interpreter. The iSBC 802 Interpreter has a complete ready-to-use set of linkable object modules which are fully compatible with Intel's iRMX 80 Real-Time Multitasking Executive Software. The modules provide interfaces to disk file and terminal I/O, software floating point, or interface to other routines provided by the user.

SPECIFICATIONS

Word Size

Instruction — 8, 16, or 24 bits

Data — 8 bits

Cycle Time

Basic Instruction Cycle — 1.95 μ sec

Note

Basic instruction cycle is defined as the fastest instruction (i.e., four clock cycles).

Memory Addressing

On-Board ROM/EPROM

0-0FFF using 2708, 2758

0-1FFF using 2716

0-3FFF using 2732

On-Board RAM

3C00-3FFF with no RAM expansion

3000-3FFF with 2114A-5 expansion

Note

All RAM configurations are automatically moved up to a base address of 4XXX when configuring EPROM for 2732.

Memory Capacity

On-Board ROM/EPROM

16K bytes (sockets only)

On-Board RAM

1K byte with user expansion in 1K increments to 4K bytes using Intel 2114A-5 RAMs

Off-Board Expansion

Up to 64K bytes using user specified combinations of RAM, ROM, and EPROM.

I/O Addressing

On-Board Programmable I/O

Device	I/O Address
8255A No. 1	
Port A	E4
Port B	E5
Port C	E6
Control	E7
8255A No. 2	
Port A	E8
Port B	E9
Port C	EA
Control	EB
8251A	
Data	EC
Control	ED
iSBX Multimodule	
MCS0	F0-F7
MCS1	F8-FF

I/O Capacity

Parallel — 48 programmable lines

Serial — 1 transmit, 1 receive

MULTIMODULE — 1 ISBX Bus MULTIMODULE Board

Serial Baud Rates

Frequency (kHz) (Jumper Selectable)	Baud Rate (Hz)	
	Synchronous	Asynchronous (Program Selectable)
307.2	—	÷ 16 ÷ 64 19200 4800
153.6	—	9600 2400
76.8	—	4800 1200
38.4	38400	2400 600
19.2	19200	1200 300
9.6	9600	600 150
6.98	6980	— 110
4.8	4800	300 75

Serial Communications Characteristics

Synchronous — 5-8 bit characters; internal or external character synchronization; automatic sync insertion

Asynchronous — 5-8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detectors

Interrupts

Single-level with on-board logic that automatically vectors the processor to location 38H using a restart instruction (RESTART7). Interrupt requests may originate from user specified I/O (2); the programmable peripheral interface (2); the ISBX MULTIMODULE board (2); the programmable communications interface (3); the power fail interrupt (1); or the interval timer (1).

Electrical Characteristics

DC Power Requirements

Voltage	Without EPROM ¹	With 2708 EPROM ²	With 2758, 2716, or 2732 EPROM ³	Power Down Requirements (RAM and Support Circuit)
V _{CC} = +5V ±5%	I _{CC} ⁴ = 2.0A	3.1 A	3.46 A	84 mA + 140 mA/K (2114A-5)
V _{DD} = +12V ±5%	I _{DD} = 150 mA	400 mA	150 mA	Not Required
V _{BB} = -5V ±5%	I _{BB} = 2 mA	200 mA	2 mA	Not Required
V _{AA} = -12V ±5%	I _{AA} = 175 mA	175 mA	175 mA	Not Required

NOTES:

- Does not include power required for optional ROM/EPROM, I/O drivers, or I/O terminators.
- With four Intel 2708 EPROMs and 220Ω/330Ω for terminators, installed for 48 input lines. All terminator inputs low.
- Same as #2 except with four 2758s, 2716s, or 2732s installed.
- I_{CC} shown without RAM supply current. For 2114A-5 add 140 mA per K byte to a maximum of 560 mA.

MULTIBUS — All signals TTL compatible

ISBX Bus — All signals TTL compatible

Parallel I/O — All signals TTL compatible

Serial I/O — RS232C or a 20 mil current loop TTY interface (jumper selectable)

Interrupt Requests — All TTL compatible (active-low)

Clocks

System Clock — 2.048 MHz ± 0.1%

Interval Timer — 1.042 msec ± 0.1% (959.5 HZ)

Connectors

Interface	Double-Sided Pins (qty)	Centers (in.)	Mating Connectors
MULTIBUS System	86	0.156	Viking 2KH43/9AMK12 Wire-wrap
ISBX Bus	36	0.1	ISBX 960-5
Parallel I/O (2)	50	0.1	3M 3415-000 Flat
Serial I/O	26	0.1	AMP 87194-6 Flat

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.05 in. (1.27 cm)

Weight — 14 oz. (397.3 gm)

Line Drivers and Terminators

I/O Drivers — The following line drivers and terminators are all compatible with the I/O driver sockets on the iSBC 80/10B Board:

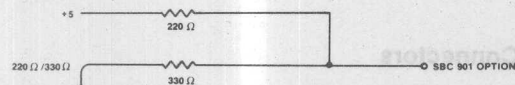
Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

Note

I - inverting, NI - non-inverting, OC - open collector.

Port 1 has 25 nA totem pole drivers and 1 k Ω terminators.

I/O Terminators — 220 Ω /330 Ω divider or 1 k Ω pull up.



ORDERING INFORMATION

Part Number	Description
SBC 80/10B	Single Board Computer

MULTIBUS Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-State	25
Address	Tri-State	25
Commands	Tri-State	25

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Equipment Supplied

iSBC 80/10B Single Board Computer

iSBC 80/10B Schematics

Reference Manual

9803119-01 — iSBC 80/10B Single Board Computer Hardware Reference Manual (NOT SUPPLIED).

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

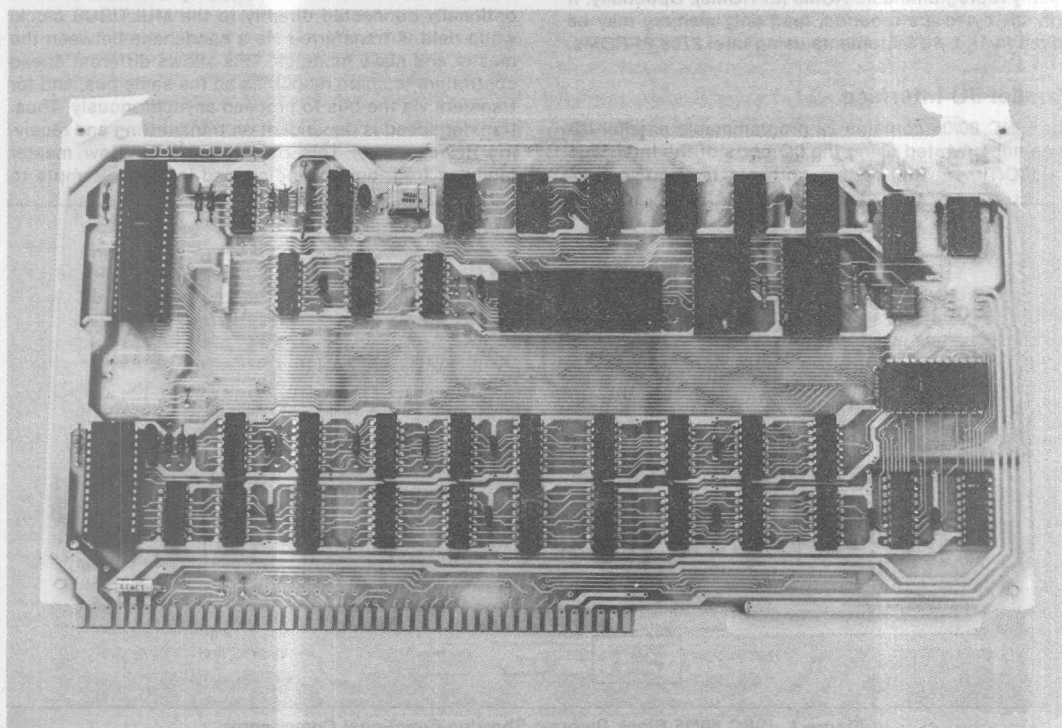


ISBC 80/05

ISBC 80/05 (or pSBC 80/05*) SINGLE BOARD COMPUTER

- 8085A CPU used as central processor
- 512 bytes of static read/write memory
- Sockets for 4K bytes of erasable reprogrammable or masked read only memory
- 22 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators
- Full MULTIBUS control logic allowing up to 16 masters to share system bus
- Programmable 14-bit binary timer
- TTL serial I/O interface with sockets for RS232C line drivers and receivers
- Four-level vectored interrupt
- Fully compatible with optional ISBC expansion boards and peripherals
- Single +5V power supply

The ISBC 80/05 Single Board Computer is a member of Intel's complete line of OEM computer systems which take full advantage of Intel's LSI technology to provide economical, self-contained computer-based solutions for OEM applications. The ISBC 80/05 is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, read/write memory, nonvolatile read only memory, I/O ports and drivers, serial interface, priority interrupt logic, programmable timer, MULTIBUS control logic, and bus expansion buffers all reside on the board.



*Same product, manufactured by Intel Puerto Rico, Inc.

FUNCTIONAL DESCRIPTION

Intel's powerful 8-bit n-channel 8085 CPU, fabricated on a single LSI chip, is the central processor for the iSBC 80/05. The 8085A CPU is directly software compatible with the popular Intel 8080A CPU. The 8085A contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators. Minimum on-board instruction execution time is 2.03 microseconds. A block diagram of iSBC 80/05 functional components is shown in Figure 1.

Memory Addressing

The 8085A CPU has a 16-bit program counter which allows direct addressing of up to 65,536 bytes of memory. An external stack, located within any portion of read/write memory, may be used as a last-in/first-out storage area for the contents of the program counter, flags, accumulator, and all of the six general purpose registers. A 16-bit stack pointer controls the addressing of this external stack. This stack provides subroutine nesting bounded only by memory size.

Memory Capacity

The iSBC 80/05 contains 512 bytes of read/write memory using Intel's low power static RAMs. Two sockets for up to 4K bytes of nonvolatile read only memory are provided on the board. Read only memory may be added in 2K-byte increments using Intel 2716 erasable and electrically reprogrammable ROMs (EPROMs). Optionally, if only 2K bytes are required, read only memory may be added in 1K-byte increments using Intel 2708 EPROMs.

Parallel I/O Interface

The iSBC 80/05 contains 22 programmable parallel I/O lines implemented using the I/O ports of the Intel 8155 RAM/I/O/Timer. The system software is used to con-

figure the I/O lines in any combination of unidirectional input or output ports as indicated in Table 1. The I/O interface may, therefore, be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 22 programmable I/O lines and signal ground lines are brought out to a 40-pin edge connector that mates with flat, woven, or round cable.

Multimaster Capability

The iSBC 8085A is a full computer on a single board with resources capable of supporting a great variety of OEM system requirements. For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically share systems tasks with communication over the system bus), the iSBC 80/05 provides full MULTIBUS arbitration control logic. This control logic allows up to three bus masters (i.e., any combination of iSBC 80/05, iSBC 80/20-4, DMA controller, diskette controller, etc.) to share the system bus in serial (daisy-chain) priority fashion, and up to 16 masters may share the MULTIBUS with the addition of an external priority network. The MULTIBUS arbitration logic operates synchronously with a MULTIBUS clock (provided by the iSBC 80/05 or optionally connected directly to the MULTIBUS clock) while data is transferred via a handshake between the master and slave modules. This allows different speed controllers to share resources on the same bus, and for transfers via the bus to proceed asynchronously. Thus, transfer speed is dependent on transmitting and receiving devices only. This design prevents slow master modules from being handicapped in their attempts to

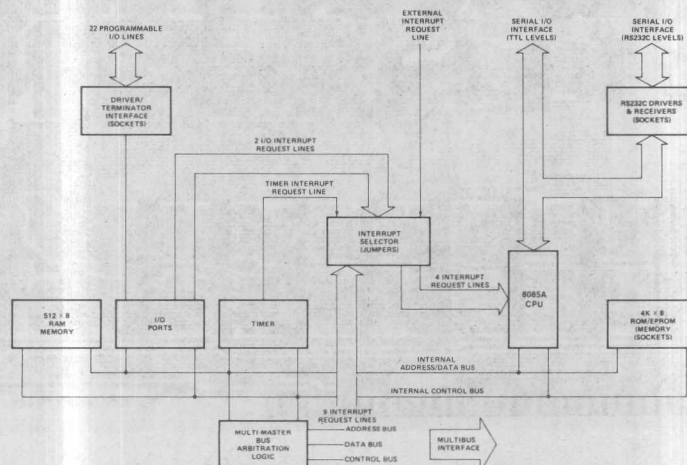


Figure 1. iSBC 80/05 Block Diagram Showing Functional Components

Table 1. Input/Output Modes of Operation

Port	Lines (qty)	Mode of Operation				Control
		Unidirectional				
		Input		Output		
		Unlatched	Latched & Strobed	Latched	Latched & Strobed	
1	8	X	X	X	X	
2	8	X	X	X	X	
3	3	X		X		X ¹
4	3	X		X		X ²

Notes

- Port 3 must be used as a control port when port 1 is used as a latched and strobed input or a latched and strobed output port.
- Port 4 must be used as a control port when port 2 is used as a latched and strobed input or a latched and strobed output port.

gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed direct-memory-access (DMA) operations and high speed peripheral control, but are by no means limited to these three.

Programmable Timer

The iSBC 80/05 provides a fully programmable binary 14-bit interval timer utilizing the Intel 8155 RAM/I/O Timer. The system designer simply configures the timer via software to meet system requirements. Whenever a given time delay is needed, software commands to the programmable timer select the desired function. Four functions are available as shown in Table 2. The contents of the timer counter may be read at any time during system operation.

Serial I/O Interface

The iSBC 80/05 provides serial I/O capability through the serial input data (SID) and serial output data (SOD) functions of the Intel 8085A CPU. These functions are controlled exclusively by software through execution of the 8085A RIM and SIM instructions. The baud rate for the serial I/O interface is determined by the system time available for execution of serial I/O support software. Hence, the maximum baud rate supported by the iSBC 80/05 is solely dependent on the overall system real-time software requirements. Serial I/O signals are TTL compatible and sockets are provided on the board for optional connection of RS232C line drivers and receivers.

Interrupt Capability

The iSBC 80/05 takes advantage of the powerful interrupt processing capability of the 8085A CPU. Interrupt requests are routed to the four interrupt inputs of the 8085A CPU (i.e., TRAP, RST 7.5, RST 6.5, and RST 5.5 in order of priority, TRAP highest), and each input generates a unique memory address (i.e., TRAP: 24₁₆, RST 7.5: 3C₁₆, RST 6.5: 34₁₆, RST 5.5: 2C₁₆). A single 8085A jump

Table 2. Programmable Timer Functions

Function	Operation
Programmable pulse	Timer out goes low during the second half of count. Therefore, the count loaded in the count length register should be twice the pulse width desired.
Square wave rate generator	Timer out will remain high until one-half the count has been completed, and go low for the other half of the count. The count length is automatically reloaded when terminal count is reached.
Rate generator	Divide by N counter. A repetitive timer out low pulse is generated and new timeout initiated every time terminal count is reached.
Programmable strobe	A single low pulse is generated upon reaching terminal count. This function is extremely useful for generation of real-time clocks.

instruction at each of these addresses then provides linkage to locate each interrupt service routine independently anywhere in memory. All interrupt inputs with the exception of one (TRAP) may be masked via software. The trap interrupt should be used for conditions such as power-down sequences which require immediate attention by the 8085A CPU.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. High speed integer and floating-point arithmetic capabilities may be added by using the iSBC 310A High Speed Mathematics Unit. Memory may be expanded to 65,536 bytes by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding

single or double density diskette controllers as sub-systems. Modular expandable backplanes and card-cages are available to support multiboard systems.

Systems Development Capability

The development cycle of ISBC 80/05-based products may be significantly reduced using Intel's system development tools available today. For those not requiring hardware emulation capability, Intel provides a new low cost microcomputer development system. The iPDS, Personal Development System, provides low cost system development for the ISBC 80/05 board, while at the same time providing personal computer capability for the engineer. The Intellec Series II family of compatible microcomputer development systems provides a range of capability from a low cost disk-

based edit debug workstation to a high performance, fully compatible hard-disk-based software development system. A unique in-circuit emulator (ICE-85A) option provides the capability of developing and debugging software directly on the ISBC 80/05 board.

Programming Capability

PL/M-80 — Intel's high level programming language, PL/M, is also available as a resident Intellec microcomputer development system option. PL/M provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M programs can be written in a much shorter time than assembly language programs for a given application.

and SIM instructions of the 8085A CPU. Baud rate is determined by system time available for serial I/O handling. On-board timer may be used to greatly ease serial I/O timing requirements.

Interrupts

Four-level interrupt routed to 8085A CPU interrupt inputs. Each interrupt automatically vectors the processor to a unique memory location.

Interrupt Input	Memory Address	Priority	Type
TRAP	24 ₁₆	Highest	Non-maskable
RST 7.5	3C ₁₆	↑ ↓	Maskable
RST 6.5	34 ₁₆		Maskable
RST 5.5	2C ₁₆		Maskable
		Lowest	

Timer

Input Frequency Reference — 122.88 kHz \pm 0.1% (8.14 μ s period nominal)

Output Frequencies/Timing Intervals

Function	Timer/Counter	
	Min	Max
Programmable pulse	8.14 μ s	66.67 ms
Square wave rate generator	7.50 Hz	61.44 kHz
Rate generator	7.50 Hz	61.44 kHz
Programmable strobe	8.14 μ s	133.33 ms

Interfaces

Bus — All signals TTL compatible

Parallel I/O — All signals TTL compatible

Interrupt Request — All TTL compatible (active-low)

Serial I/O — TTL; sockets available for RS232C line drivers and receivers

System Clock (8085A CPU)

1.966 MHz \pm 0.1%

SPECIFICATIONS

Word Size

Instruction — 8, 16, or 24 bits

Data — 8 bits

Cycle Time

Basic Instruction Cycle — 2.03 μ s, \pm 0.1%

Note

Basic instruction cycle is defined as the fastest instruction (i.e., four clock cycles).

Memory Addressing

ROM/EPROM — 0-0FFF_H

RAM — 3E00_H

Memory Capacity

On-Board ROM/EPROM — 4K bytes (with Intel 2716) or 2K bytes (with Intel 2708)

On-Board RAM — 512 bytes

Off-Board Expansion — Up to 65,536 bytes in user specified combination of RAM, ROM, and PROM

I/O Addressing

On-Board Programmable I/O — see Table 1

Port Control	8155 Port 1	8155 Port 2	8155 Ports 3 & 4	8155 Port	8155 Timer Low-Order Byte	8155 Timer High-Order Byte
Address	00	01	02	03	04	05

I/O Capacity

Parallel — 22 programmable lines (see Table 1)

Note

The ISBC 80/05 may be expanded to 1102 programmable input/output lines by using optional ISBC 80 I/O boards.

Serial Communications Characteristics

SID and SOD functions of the 8085A CPU are used for serial I/O. They are controlled by software through RIM

Interface	Lines (qty)	Centers (in.)	Mating Connector
Bus	86 double-sided	0.156	Viking 2KH43/9AMK12
Parallel I/O	50 double-sided	0.100	3M 3415-000
Serial I/O ¹	7 single-sided	0.156	Molex 09-66-1071 Connector Molex 09-50-7071 Connector
			AMP 87194-6 Connector AMP 3-87025-4 Connector

Note

1. Connectors and pins from one vendor may only be used with connectors and pins from the same vendor.

Line Drivers and Terminators

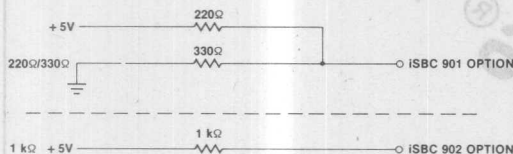
I/O Drivers — The following line drivers are all compatible with the I/O driver sockets on the iSBC 80/05:

Driver	Characteristic	Sink Current (mA)
7438	I,OC	48
7437	I	48
7432	NI	16
7426	I,OC	16
7409	NI,OC	16
7408	NI	16
7403	I,OC	16
7400	I	16

Note

I = inverting; NI = non-inverting; OC = open collector.

I/O Terminators — Intel provides 220 Ω /330 Ω divider and 1 k Ω pull-up resistive terminator packs for termination of I/O lines programmed as inputs. These options are as follows:

**Bus Drivers**

Driver	Characteristic	Sink Current (mA)
Data	Tri-state	50
Address	Tri-state	50
Commands	Tri-state	32

RS232C Drivers and Receivers

The following RS232C drivers and receivers are compatible with the RS232C socket on the iSBC 80/05:

RS232C Driver — National DS1488 or TI SN75188

RS232C Receiver — National DS1490 or TI SN75189

Physical Characteristics

Width — 12.00 in. (30.49 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.50 in. (1.27 cm)

Weight — 12.0 oz (339.8 gm)

Electrical Characteristics**DC Power Requirements**

Voltage ($\pm 5\%$)	Without PROM ¹ (max)	With 2716 EPROM ² (max)	With 8708 EPROM ³ (max)
$V_{CC} = +5V$	$I_{CC} = 1.80 \text{ mA}$	2.65A	2.45A
$V_{DD} = +12V$ ⁴	$I_{DD} = 0$	7 mA ⁵	137 mA
$V_{BB} = -5V$ ⁴	$I_{BB} = 0$	0	90 mA
$V_{AA} = -12V$ ⁵	$I_{AA} = 0$	23 mA ⁵	23 mA ⁵

Notes

- Does not include power required for optional EPROM/ROM, I/O drivers, and I/O terminators.
- With two Intel 2716 EPROMs and 220 Ω /330 Ω terminators installed for 22 input ports; all terminator inputs low.
- With two Intel 2708 EPROMs and 220 Ω /330 Ω terminators installed for 22 input ports; all terminator inputs low.
- Required for 2708 EPROMs.
- Required only when RS232C capability required.

Environmental Characteristics

Operating Temperature — 0°C to +55°C.

Reference Manual

9800483D — iSBC 80/05 Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 80/05	Single Board Computer

I. INTRODUCTION

A significant measure of the power and flexibility of the Intel OEM Computer Product Line can be attributed to the design of the Intel MULTIBUS system bus. The bus structure provides a common element for communication between a wide variety of system modules which include: Single Board Computers, memory, digital, and analog I/O expansion boards, and peripheral controllers.

The purpose of this application note is to help you develop a working knowledge of the Intel MULTIBUS specification. This knowledge is essential for configuring a system containing multiple modules. Another purpose is to provide you with the information necessary to design a bus interface for a slave module. One of the tools that will be used to achieve this goal is the complete description of a MULTIBUS slave design example. Other portions of this application note provide an in depth examination of the bus signals, operating characteristics, and bus interface circuits.

This application note was originally written in 1977. Since 1977, the MULTIBUS specification has been significantly expanded to cover operation with both 8 and 16-bit system modules and with an auxiliary power bus. This application note now contains information on these new MULTIBUS specification features.

In addition, a detailed MULTIBUS specification has also been published which provides the user with further information concerning MULTIBUS interfacing. The MULTIBUS specification and other useful documents are listed in the overleaf of this note under Related Intel Publications.

II. MULTIBUS® SYSTEM BUS DESCRIPTION

Overview

The Intel MULTIBUS signal lines can be grouped in the following categories: 20 address lines, 16 bidirectional data lines, 8 multilevel interrupt lines, and several bus control, timing and power supply lines. The address and data lines are driven by three-state devices, while the interrupt and some other control lines are open-collector driven.

Modules that use the MULTIBUS system bus have a master-slave relationship. A bus master module can drive the command and address lines: it can control the bus. A Single Board Computer is an example of a bus master. A bus slave cannot

control the bus. Memory and I/O expansion boards are examples of bus slaves. The MULTIBUS architecture provides for both 8 and 16-bit bus masters and slaves.

Notice that a system may have a number of bus masters. Bus arbitration results when more than one master requests control of the bus at the same time. A bus clock is usually provided by one of the bus masters and may be derived independently from the processor clock. The bus clock provides a timing reference for resolving bus contention among multiple requests from bus masters. For example, a processor and a DMA (direct memory access) module may both request control of the bus. This feature allows different speed masters to share resources on the same bus. Actual transfers via the bus, however, proceed asynchronously with respect to the bus clock. Thus, the transfer speed is dependent on the transmitting and receiving devices only. The bus design prevents slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. Once a bus request is granted, single or multiple read/write transfers can proceed. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations and high-speed direct-memory-access (DMA) operations. However, the master-slave capabilities of the bus are by no means limited to these two applications.

MULTIBUS® Signal Descriptions

This section defines the signal lines that comprise the Intel MULTIBUS system bus. These signals are contained on either the P1 or P2 connector of boards compatible with the MULTIBUS specification. The P1 signal lines contain the address, data, bus control, bus exchange, interrupt and power supply lines. The P2 signal lines contain the optional auxiliary signal lines. Most signals on the bus are active-low. For example, a low level on a control signal on the bus indicates active, while a low level on an address or data signal on the bus represents logic "1" value.

NOTE

In this application note, a signal will be designated active-low by placing a slash (/) after the mnemonic for the signal.

Appendix A contains a pin assignment list of the following signals:

MULTIBUS P1 Signal Lines —

INIT/

Initialization signal; resets the entire system to a known internal state. INIT/ may be driven by one of the bus masters or by an external source such as a front panel reset switch.

Address and Inhibit Lines

20 address lines/ used to transmit the address of the memory location or I/O port to be accessed. The lines are labeled ADR0/ through ADR9/ ADR10/ through ADR13/ for memory addressing and 12 address lines (ADR0/ - ADR7/) for I/O port selection. 8-bit masters use 16 address lines (ADR0/ - ADR7/) for memory addressing and 8 address lines (ADR8/ - ADR13/) for I/O port selection. 16-bit masters use all twenty address lines for memory addressing and 12 address lines (ADR0/ - ADR11/) for I/O port selection. Thus, 8-bit masters may address 64K bytes of memory and 256 I/O devices while 16-bit masters may address 1 megabyte of memory and 4096 I/O devices. (The 8086 CPU actually permits 16 address bits to be used to specify I/O devices, the MULTIBUS specification, however, states that only the low order 12 address bits can be used to specify I/O ports.) In a 16-bit system, the ADR0/ line is used to indicate whether a low (even) byte or a high (odd) byte of memory or I/O space is being accessed in a word oriented memory or I/O device.

BHEN/

Byte High Enable; the address control line which is used to specify that data will be transferred on the high byte (DAT8/- DATF/) of the MULTIBUS data lines. With current iSBC boards, this signal effectively specifies that a word (two byte) transfer is to be performed. This signal is used only in systems which incorporate sixteen bit memory or I/O modules.

INH1/

Inhibit RAM signal; prevents RAM memory devices from responding to the memory address on the system address bus. INH1/ effectively allows ROM memory devices to override RAM devices when ROM and RAM memory are

assigned the same memory addresses. INH1/ may also be used to allow memory mapped I/O devices to override RAM memory.

INH2/

Inhibit ROM signal; prevents ROM memory devices from responding to the memory address on the system address bus. INH2/ effectively allows auxiliary ROM (e.g., a bootstrap program) to override ROM devices when ROM and auxiliary ROM memory are assigned the same memory addresses. INH2/ may also be used to allow memory mapped I/O devices to override ROM memory.

Data Lines

DAT0/ - DATE/

16 bidirectional data lines; used to transmit or receive information to or from a memory location or I/O port. DATF/ being the most significant bit. In 8-bit systems, only lines DAT0/ - DAT7/ are used (DAT7/ being the most significant bit). In 16-bit systems, either 8 or 16 lines may be used for data transmission.

Bus Priority Resolution Lines

BCLK/

Bus clock; the negative edge (high to low) of BCLK/ is used to synchronize bus priority resolution circuits. BCLK/ is asynchronous to the CPU clock. It has a 100 ns minimum period and a 35% to 65% duty cycle. BCLK/ may be slowed, stopped, or single stepped for debugging.

CCLK/

Constant clock; a bus signal which provides a clock signal of constant frequency for unspecified general use by modules on the system bus. CCLK/ has a minimum period of 100 ns and a 35% to 65% duty cycle.

BPRN/

Bus priority in signal; indicates to a particular master module that no higher priority module is requesting use of the system bus. BPRN/ is synchronized with BCLK/. This signal is not based on the backplane.

BPRO/

Bus priority out signal; used with serial (daisy chain) bus priority resolution schemes. BPRO/ is passed to the BPRN/ input of the master module with the next lower bus priority. BPRO/ is synchronized with BCLK/. This signal is not based on the backplane.

BUSY/

Bus busy signal; an open collector line driven by the bus master currently in control to indicate that the bus is currently in use. BUSY/ prevents all other master modules from gaining control of the bus. BUSY/ is synchronized with BCLK/.

BREQ/

Bus request signal; used with a parallel bus priority network to indicate that a particular master module requires use of the bus for one or more data transfers. BREQ/ is synchronized with BCLK/. This signal is not based on the backplane.

CBRQ/

Common bus request; an open-collector line which is driven by all potential bus masters and is used to inform the current bus master that another master wishes to use the bus. If CBRQ/ is high, it indicates to the bus master that no other master is requesting the bus, and therefore, the present bus master can retain the bus. This saves the bus exchange overhead for the current master.

Information Transfer Protocol Lines

A bus master provides separate read/write command signals for memory and I/O devices: MRDC/, MWTC/, IORC/ and IOWC/, as explained below. When a read/write command is active, the address signals must be stabilized at all slaves on the bus. For this reason, the protocol requires that a bus master must issue address signals (and data signals for a write operation) at least 50 ns ahead of issuing a read/write command to the bus, initiating the data transfer. The bus master must keep address signals unchanged until at least 50 ns after the read/write command is turned off, terminating the data transfer.

A bus slave must provide an acknowledge signal to

the bus master in response to a read or write command signal.

MRDC/

Memory read command; indicates that the address of a memory location has been placed on the system address lines and specifies that the contents (8 or 16 bits) of the addressed location are to be read and placed on the system data bus. MRDC/ is asynchronous with respect to BCLK/.

MWTC/

Memory write command; indicates that the address of a memory location has been placed on the system address lines and that data (8 or 16 bits) has been placed on the system data bus. MWTC/ specifies that the data is to be written into the addressed memory location. MWTC/ is asynchronous with respect to BCLK/.

IORC/

I/O read command; indicates that the address of an input port has been placed on the system address bus and that the data (8 or 16 bits) at that input port is to be read and placed on the system data bus. IORC/ is asynchronous with respect to BCLK/.

IOWC/

I/O write command; indicates that the address of an output port has been placed on the system address bus and that the contents of the system data bus (8 or 16 bits) are to be output to the address port. IOWC/ is asynchronous with respect to BCLK/.

XACK/

Transfer acknowledge signal; the required response of a slave board which indicates that the specified read/write operation has been completed. That is, data has been placed on, or accepted from, the system data bus lines. XACK/ is asynchronous with respect to BCLK/.

Asynchronous Interrupt Lines**INT0/-INT7/**

8 Multi-level, parallel interrupt request lines;

used with a parallel interrupt resolution network. INT0/ has the highest priority, while INT7/ has lowest priority. Interrupt lines should be driven with open collector drivers.

INTA/

Interrupt acknowledge; an interrupt acknowledge line (INTA/), driven by the bus master, requests the transfer of interrupt information onto the bus from slave priority interrupt controllers (8259s or 8259As). The specific information timed onto the bus depends upon the implementation of the interrupt scheme. In general, the leading edge of INTA/ indicates that the address bus is active while the trailing edge indicates that data is present on the data lines.

MULTIBUS P2 Signal Lines — The signals contained on the MULTIBUS P2 auxiliary connector are used primarily by optional power back-up circuitry for memory protection. P2 signals are not bused on the backplane, and therefore, require a separate connector for each board using the P2 signals. Present iSBC boards have a slot in the card edge and should be used with a keyed P2 edge connector. Use of the P2 signal lines is optional.

ACLO

AC Low; this signal generated by the power supply goes high when the AC line voltage drops below a certain voltage (e.g., 103v AC in 115v AC line voltage systems) indicating D.C. power will fail in 3 msec. ACLO goes low when all D.C. voltages return to approximately 95% of the regulated value. This line must be pulled up by the optional standby power source, if one is used.

PFIN/

Power fail interrupt; this signal interrupts the processor when a power failure occurs, it is driven by external power fail circuitry.

PFSN/

Power fail sense; this line is the output of a latch which indicates that a power failure has occurred. It is reset by PFSR/. The power fail

sense latch is part of external power fail circuitry and must be powered by the standby power source.

PFSR/

Power fail sense reset; this line is used to reset the power fail sense latch (PFSN/).

MPRO/

Memory protect; prevents memory operation during period of uncertain DC power, by inhibiting memory requests. MPRO/ is driven by external power fail circuitry.

ALE

Address latch enable; generated by the CPU (8085 or 8086) to provide an auxiliary address latch.

HALT/

Halt; indicates that the master CPU is halted.

AUX RESET/

Auxiliary Reset; this externally generated signal initiates a power-up sequence.

WAIT/

Bus master wait state; this signal indicates that the processor is in a wait state.

Reserved — Several P1 and P2 connector bus pins are unused. However, they should be regarded as reserved for dedicated use in future Intel products.

Power Supplies — The power supply bus pins are detailed in Appendix A which contains the pin assignment of signals on the MULTIBUS backplane.

It is the designer's responsibility to provide adequate bulk decoupling on the board to avoid current surges on the power supply lines. It is also recommended that you provide high frequency

decoupling for the logic on your board. Values of 22 μ F for +5v and +12v pins and 10 μ F for -5v and -12v pins are typical on iSBC boards.

Operating Characteristics

Beyond the definition of the MULTIBUS signals themselves, it is important to examine the operating characteristics of the bus. The AC requirements outline the timing of the bus signals and in particular, define the relationships between the various bus signals. On the other hand, the DC requirements specify the bus driver characteristics, maximum bus loading per board, and the pull-up/down resistors.

The AC requirements are best presented by a discussion of the relevant timing diagrams. Appendix B contains a list of the MULTIBUS timing specifications. The following sections will discuss data transfers, inhibit operations, interrupt operations, MULTIBUS multi-master operation and power fail considerations.

Data Transfers — Data transfers on the MULTIBUS system bus occur with a maximum bandwidth of 5 MHz for single or multiple read/write transfers. Due to bus arbitration and memory access time, a typical maximum transfer rate is often on the order of 2 MHz.

Read Data

Figure 1 shows the read operation AC timing diagram. The address must be stable (t_{AS}) for a minimum of 50 ns before command (IORC/ or MRDC/). This time is typically used by the bus interface to decode the address and thus provide the required device selects. The device selects establish the data paths on the user system in anticipation of the strobe signal (command) which will follow. The minimum command pulse width is 100 ns. The address must remain stable for at least 50 ns following the command (t_{AH}). Valid data should not be driven onto the bus prior to command, and must not be removed until the command is cleared. The XACK/ signal, which is a response indicating the specified read/write operation has been completed, must coincide or follow both the read access and valid data (t_{DXL}). XACK/ must be held until the command is cleared (t_{XAH}).

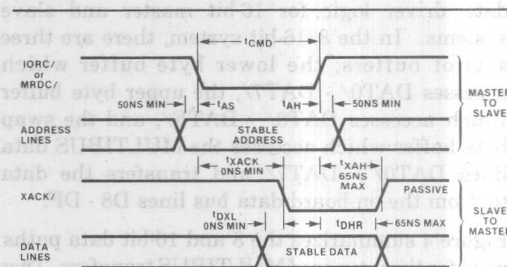


Figure 1. Read AC Timing

Write Data

The write operation AC timing diagram is shown in Figure 2. During a write data transfer, valid data must be presented simultaneously with a stable address. Thus, the write data setup time (t_{DS}) has the same requirement as the address setup time (t_{AS}). The requirement for stable data both before and after command (IOWC/ or MWTC/) enables the bus interface circuitry to latch data on either the leading or trailing edge of command.

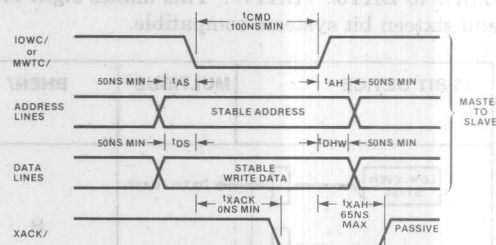


Figure 2. Write AC Timing

Data Byte Swapping in 16-bit Systems

A 16-bit master may transfer data on the MULTIBUS data lines using 8-bit or 16-bit paths depending on whether a byte or word (2 byte) operation has been specified. (A word transfer specified with an odd I/O or memory address will actually be executed as two single byte transfers.) An 8-bit master may only perform byte transfers on the MULTIBUS data lines DAT0/ - DAT7/.

In order to maintain compatibility with older 8-bit masters and slaves, a byte swapping buffer is included in all new 16-bit masters and 16-bit slaves. In the iSBC product line, all byte transfers will take place on the low 8 data lines DAT0/ - DAT7/. Figure 3 contains an example of 8/16-bit

data driver logic for 16-bit master and slave systems. In the 8/16-bit system, there are three sets of buffers; the lower byte buffer which accesses DAT0/- DAT7/, the upper byte buffer which accesses DAT8/- DATF/, and the swap byte buffer which accesses the MULTIBUS data lines DAT0/- DAT7/ and transfers the data to/from the on-board data bus lines D8 - DF.

Figure 4 summarizes the 8 and 16-bit data paths used for three types of MULTIBUS transfers. Two signals control the data transfers.

Byte High Enable (BHEN/) active indicates that the bus is operating in sixteen bit mode, and Address Bit 0 (ADRO/) defines an even or odd byte transfer address.

On the first type of transfer, BHEN/ is inactive, and ADRO/ is inactive indicating the transfer of an even eight bit byte. The transfer takes place across data lines DAT0/- DAT7/.

On the second type of transfer, BHEN/ is inactive, and ADRO/ is active indicating the transfer of a high (odd) byte. On this type of transfer, the odd (high) byte is transferred through the Swap Byte Buffer to DAT0/- DAT7/. This makes eight bit and sixteen bit systems compatible.

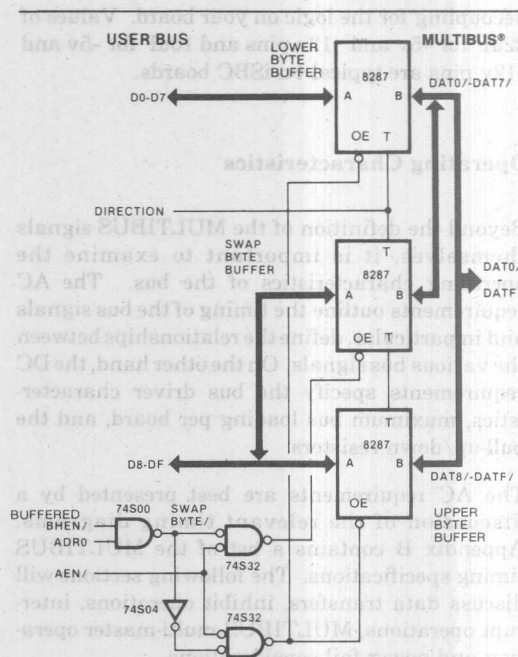


Figure 3. 8/16-Bit Data Drivers

16-BIT DEVICE	MULTIBUS®	BHEN/	ADRO/	MULTIBUS® TRANSFER DATA PATH	DEVICE BYTE TRANSFERRED
		H	H	8-BIT, DAT0/- DAT7/	EVEN
		H	L	8-BIT, DAT0/- DAT7/	ODD
		L	H	16-BIT, DAT0/- DATF/	EVEN AND ODD

Figure 4. 8/16-Bit Device Transfer Operation

The third type of transfer is a 16 bit (word) transfer. This is indicated by BHEN/ being active, and ADR0/ being inactive. On this type of transfer, the low (even) byte is transferred on DAT0/ - DAT7/ and the high (odd) byte is transferred on DAT8/ - DATF/.

Note that the condition when both BHEN/ and ADR0/ are active is not used with present iSBC boards. This condition could be used to transfer a high odd byte of data on DAT8/ - DATF/, thus eliminating the need for the swap byte buffer. However, this is not a recommended transfer type, because it eliminates the capability of communicating with 8-bit modules.

Inhibit Operations — Bus inhibit operations are required by certain bootstrap and memory mapped I/O configurations. The purpose of the inhibit operation is to allow a combination of RAM, ROM, or memory mapped I/O to occupy the same memory address space. In the case of a bootstrap, it may be desirable to have both ROM and RAM memory occupy the same address space, selecting ROM instead of RAM for low order memory only when the system is reset. A system designed to use

memory mapped I/O, which has actual memory occupying the memory mapped I/O address space, may need to inhibit RAM or ROM memory to perform its functions.

There are two essential requirements for a successful inhibit operation. The first is that the inhibit signal must be asserted as soon as possible, within a maximum of 100 ns (t_{CI}), after stable address. The second requirement for a successful inhibit operation is that the acknowledge must be delayed (t_{XACKB}) to allow the inhibited slave to terminate any irreversible timing operations initiated by detection of a valid command prior to its inhibit.

This situation may arise because a command can be asserted within 50 ns after stable address (t_{AS}) and yet inhibit is not required until 100 ns (t_{ID}) after stable address. The acknowledge delay time (t_{XACKB}) is a function of the cycle time of the inhibited slave memory. Inhibiting the iSBC 016 RAM board, for example, requires a minimum of 1.5 μ sec. Less time is typically needed to inhibit other memory modules. For example, the iSBC 104 board requires 475 ns.

Figure 5 depicts a situation in which both RAM

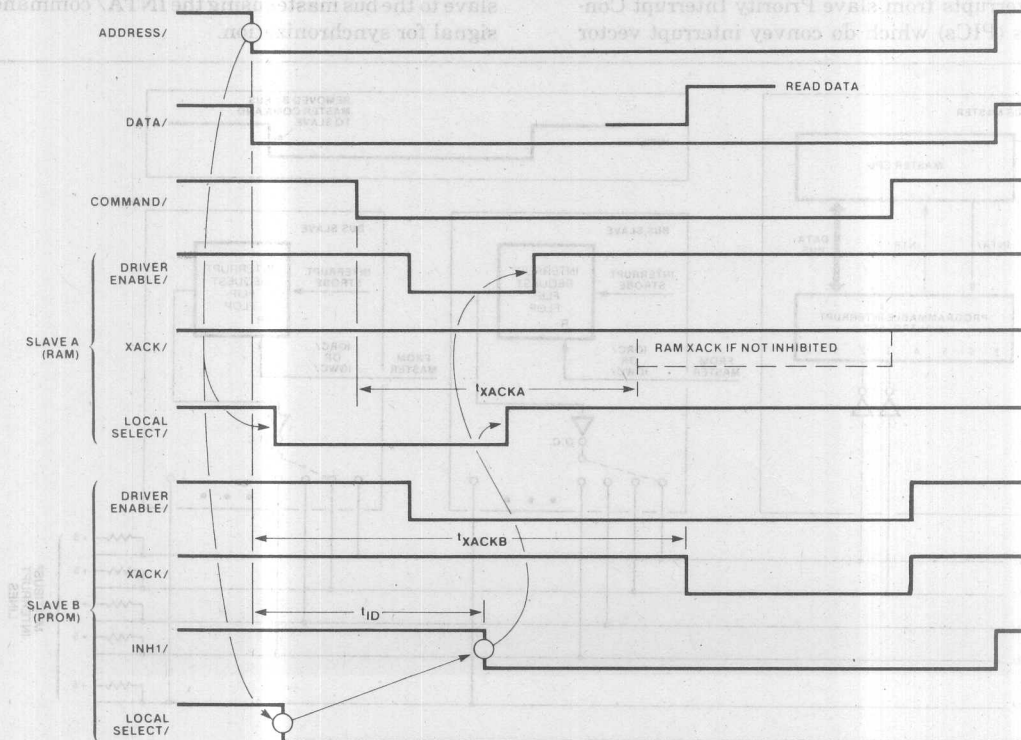


Figure 5. Inhibit Timing

and PROM memory have the same memory addresses. In this case, PROM inhibits RAM, producing the effect of PROM overriding RAM. After address is stable, local selects are generated for both the PROM and the RAM. The PROM local select produces the INH1/ signal which then removes the RAM local select and its driver enable. Because the slave RAM has been inhibited after it had already begun its cycle, the PROM XACK/ must be delayed (tXACKB) until after the latest possible acknowledgement from the RAM (tXACKA).

Interrupt Operations — The MULTIBUS interrupt lines INT0/ - INT7/ are used by a MULTIBUS master to receive interrupts from bus slaves, other bus masters or external logic such as power fail logic. A bus master may also contain internal interrupt sources which do not require the bus interrupt lines to interrupt the master. There are two interrupt implementation schemes used by bus interrupts, Non Bus Vectored Interrupts and Bus Vectored Interrupts. Non Bus Vectored Interrupts do not convey interrupt vector address information on the bus. Bus Vectored Interrupts are interrupts from slave Priority Interrupt Controllers (PICs) which do convey interrupt vector

address information on the bus.

Non Bus Vectored Interrupts

Non Bus Vectored Interrupts are those interrupts whose interrupt vector address is generated by the bus master and do not require the MULTIBUS address lines for transfer of the interrupt vector address. The interrupt vector address is generated by the interrupt controller on the master and transferred to the processor over the local bus. The source of the interrupt can be on the master module or on other bus modules, in which case the bus modules use the MULTIBUS interrupt request lines (INT0/ - INT7/) to generate their interrupt requests to the bus master. When an interrupt request line is activated, the bus master performs its own interrupt operation and processes the interrupt. Figure 6 shows an example of Non Bus Vectored Interrupt implementation.

Bus Vectored Interrupts

Bus Vectored Interrupts (Figure 7) are those interrupts which transfer the interrupt vector address along the MULTIBUS address lines from the slave to the bus master using the INTA/ command signal for synchronization.

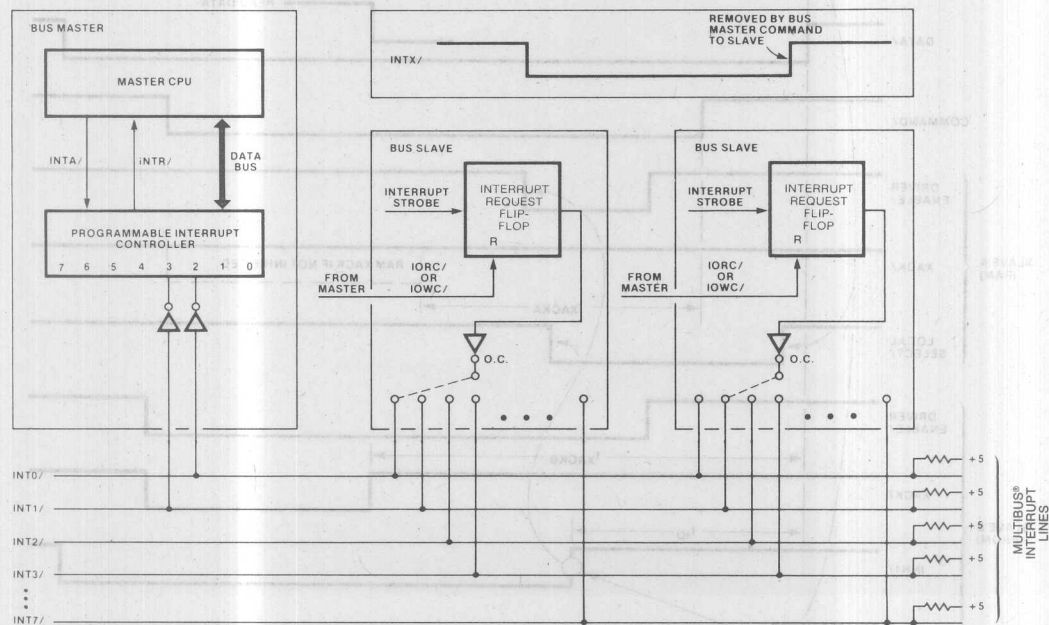


Figure 6. Non Bus Vectored Interrupt Implementation

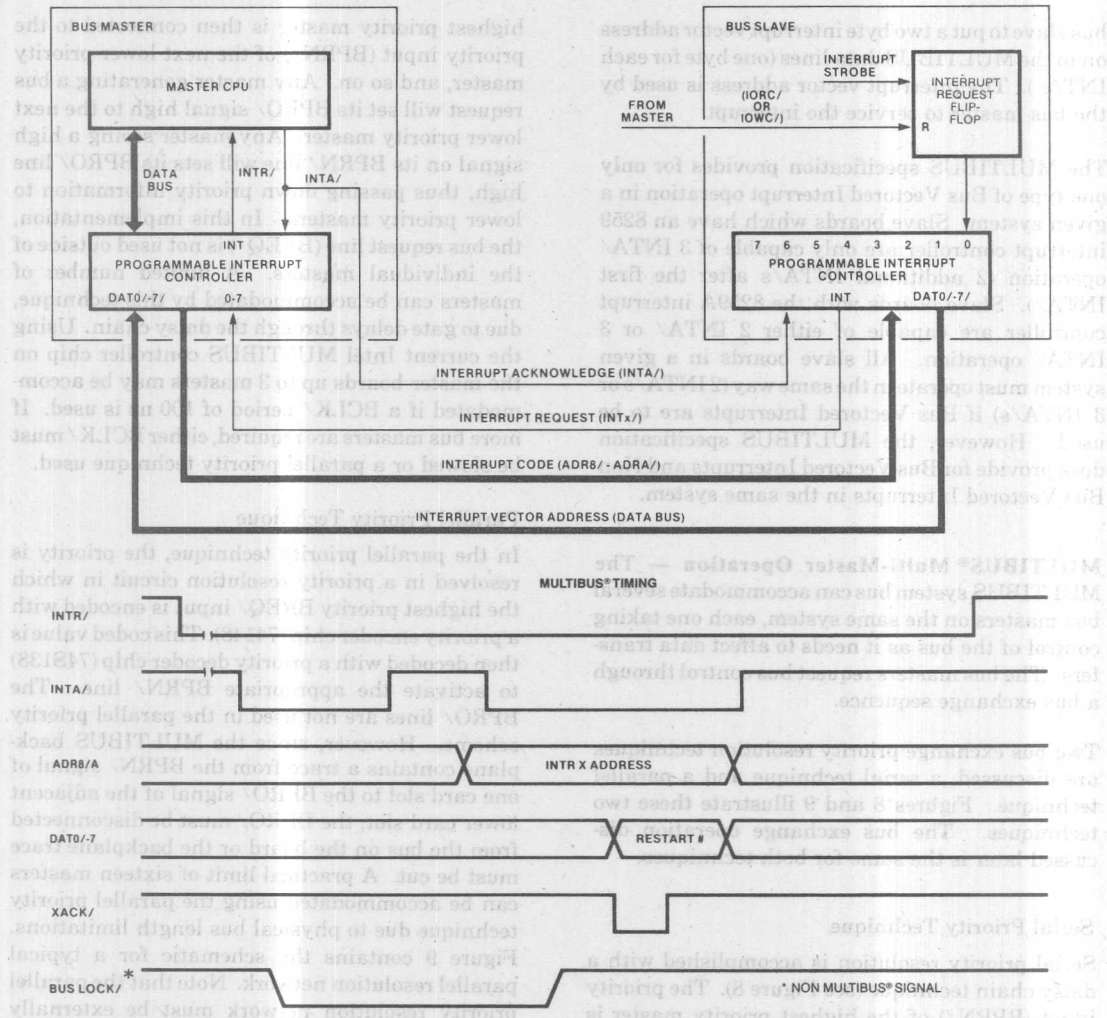


Figure 7. Bus Vectored Interrupt Logic (With 2 INTA/ Timing Diagram)

When an interrupt request from the MULTIBUS interrupt lines INT0/ - INT7/ occurs, the interrupt control logic on the bus master interrupts its processor. The processor on the bus master generates an INTA/ command which freezes the state of the interrupt logic on the MULTIBUS slaves for priority resolution. The bus master also locks (retains the bus between bus cycles) the MULTIBUS control lines to guarantee itself consecutive bus cycles. After the first INTA/ command, the bus master's interrupt control logic puts an interrupt code on to the MULTIBUS address lines ADR8/ - ADRA/. The interrupt code is the address of the highest priority active interrupt request line. At this point in the Bus Vectored

Interrupt procedure, two different sequences could take place. The difference occurs, because the MULTIBUS specification can support masters which generate one additional INTA/ (8086 masters) or two additional INTA/s (8080A and 8085 masters).

If the bus master generates one additional INTA/, this second INTA/ causes the bus slave interrupt control logic to transmit an interrupt vector 8-bit pointer on the MULTIBUS data lines. The vector pointer is used by the bus master to determine the memory address of the interrupt service routine.

If the bus master generates two additional INTA/s, these two INTA/ commands allow the

bus slave to put a two byte interrupt vector address on to the MULTIBUS data lines (one byte for each INTA/). The interrupt vector address is used by the bus master to service the interrupt.

The MULTIBUS specification provides for only one type of Bus Vectored Interrupt operation in a given system. Slave boards which have an 8259 interrupt controller are only capable of 3 INTA/ operation (2 additional INTA/s after the first INTA/). Slave boards with the 8259A interrupt controller are capable of either 2 INTA/ or 3 INTA/ operation. All slave boards in a given system must operate in the same way (2 INTA/s or 3 INTA/s) if Bus Vectored Interrupts are to be used. However, the MULTIBUS specification does provide for Bus Vectored Interrupts and Non Bus Vectored Interrupts in the same system.

MULTIBUS® Multi-Master Operation — The MULTIBUS system bus can accommodate several bus masters on the same system, each one taking control of the bus as it needs to affect data transfers. The bus masters request bus control through a bus exchange sequence.

Two bus exchange priority resolution techniques are discussed, a serial technique and a parallel technique. Figures 8 and 9 illustrate these two techniques. The bus exchange operation discussed later is the same for both techniques.

Serial Priority Technique

Serial priority resolution is accomplished with a daisy chain technique (see Figure 8). The priority input (BPRN/) of the highest priority master is tied to ground. The priority output (BPRO/) of the

highest priority master is then connected to the priority input (BPRN/) of the next lower priority master, and so on. Any master generating a bus request will set its BPRO/ signal high to the next lower priority master. Any master seeing a high signal on its BPRN/ line will set its BPRO/ line high, thus passing down priority information to lower priority masters. In this implementation, the bus request line (BREQ/) is not used outside of the individual masters. A limited number of masters can be accommodated by this technique, due to gate delays through the daisy chain. Using the current Intel MULTIBUS controller chip on the master boards up to 3 masters may be accommodated if a BCLK/ period of 100 ns is used. If more bus masters are required, either BCLK/ must be slowed or a parallel priority technique used.

Parallel Priority Technique

In the parallel priority technique, the priority is resolved in a priority resolution circuit in which the highest priority BREQ/ input is encoded with a priority encoder chip (74148). This coded value is then decoded with a priority decoder chip (74S138) to activate the appropriate BPRN/ line. The BPRO/ lines are not used in the parallel priority scheme. However, since the MULTIBUS backplane contains a trace from the BPRN/ signal of one card slot to the BPRO/ signal of the adjacent lower card slot, the BPRO/ must be disconnected from the bus on the board or the backplane trace must be cut. A practical limit of sixteen masters can be accommodated using the parallel priority technique due to physical bus length limitations. Figure 9 contains the schematic for a typical parallel resolution network. Note that the parallel priority resolution network must be externally supplied.

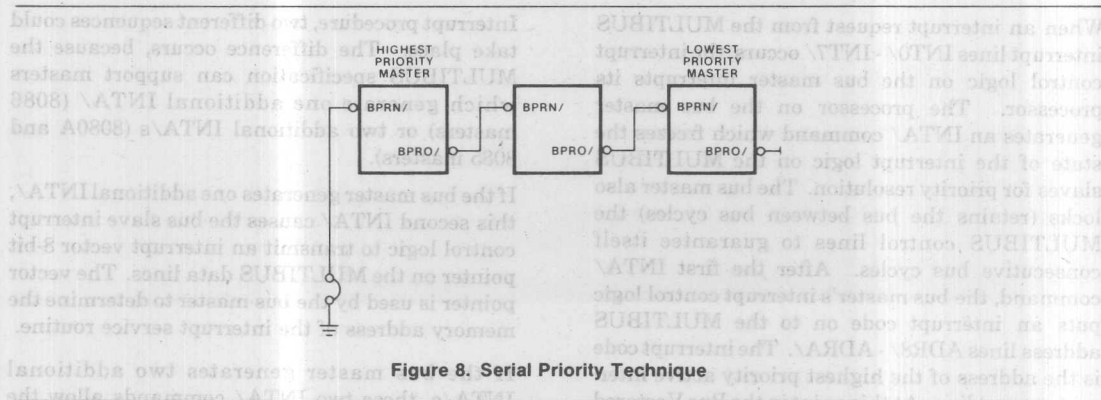


Figure 8. Serial Priority Technique

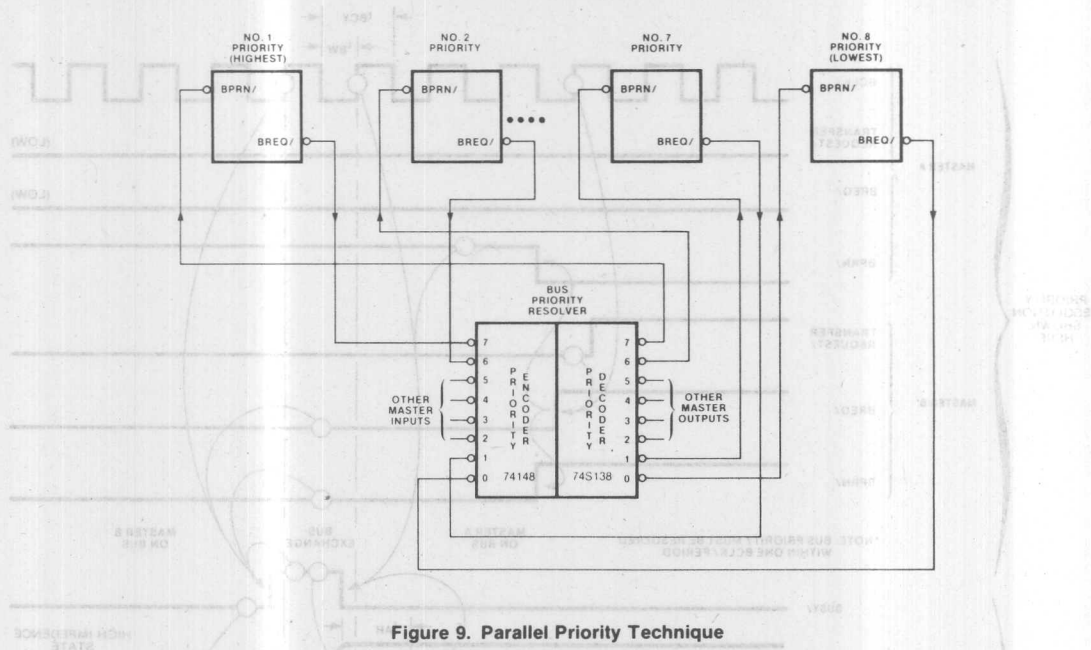


Figure 9. Parallel Priority Technique

MULTIBUS® Exchange Operation — A timing diagram for the MULTIBUS exchange operation is shown in Figure 10. This implementation example uses a parallel resolution scheme, however, the timing would be basically the same for the serial resolution scheme.

In this example, master A has been assigned a lower priority than master B. The bus exchange occurs because master B generates a bus request during a time when master A has control of the bus.

The exchange process begins when master B requires the bus to access some resource such as an I/O or memory module while master A controls the bus. This internal request is synchronized with the trailing edge (high to low) of BCLK/ to generate a bus request (BREQ/). The bus priority resolution circuit changes the BPRN/ signal from active (low) to inactive (high) for master A and from inactive to active for master B. Master A must first complete the current bus command if one is in operation. After master A completes the command, it sets BUSY/ inactive on the next trailing edge of BCLK/. This allows the actual bus exchange to occur, because master A has relinquished control of the bus, and master B has been granted its BPRN/. During this time, the drivers

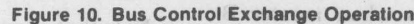
for master A are disabled. Master B must take control of the bus with the next trailing edge of BCLK/ to complete the bus exchange. Master B takes control by activating BUSY/ and enabling its drivers.

It is possible for master A to retain control of the bus and prevent master B from getting control. Master A activates the Bus Override (or Bus Lock) signal which keeps BUSY/ active allowing control of the bus to stay with master A. This guarantees a master consecutive bus cycles for software or hardware functions which require exclusive, continuous access to the bus.

Note that in systems with only a single master it is necessary to ground the BPRN/ pin of the master, if slave boards are to be accessed. In single board systems which use a CPU board capable of Bus Vectored Interrupt operation, the BPRN/ pin must also be grounded.

In a single master system bus transfer efficiency may be gained if the BUS OVERRIDE signal is kept active continuously. This permits the master to maintain control of the bus at all times, therefore saving the overhead of the master reacquiring the bus each time it is needed.

The CBRQ/ line may be used by a master in control of the bus to determine if another master



Note that except for the BUS OVERRIDE state, no single master may keep exclusive control of the bus. This is true because it is impossible for the CPU on a master to require continuous access to the bus. Other lower priority masters will always be able to gain access to the bus between accesses of a higher priority master.

Power Fail Considerations—The MULTIBUS P2 connector signals provide a means of handling power failures. The circuits required for power

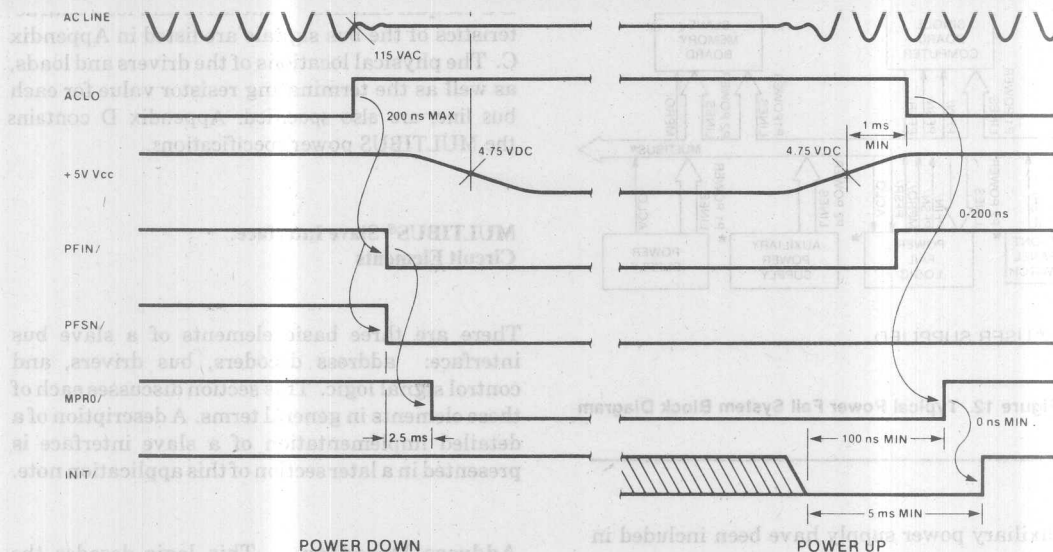


Figure 11. Power Fail Timing Sequence

failure detection and handling are optional and must be supplied by the user. Figure 11 shows the timing of a power fail sequence.

The power supply monitors the AC power level. When power drops below an acceptable value, the power supply raises ACLO which tells the power fail logic that a minimum of three milliseconds will elapse before DC power will fall below regulated voltage levels. The power fail logic sets a sense latch (PFSN/) and generates an interrupt (PFIN/) to the processor so the processor can store its environment. After a 2.5 millisecond timeout, the memory protect signal (MPRO/) is asserted by the power fail logic preventing any memory activity. As power falls, the memory goes on standby power. Note that the power fail logic must be powered from the standby source.

As the AC line revives, the logic voltage level is monitored by the power supply. After power has been at its operating level for one millisecond minimum, the power supply sets the signal ACLO low, beginning the restart sequence. First, the memory protect line (MPRO/) then the initialize line (INIT/) become inactive. The bus master now starts running. The bus master checks the power fail latch (PFSN/) and, if it finds it set, branches to

a power up routine which resets the latch (PFSN/), restores the environment, and resumes execution.

Note that INIT/ is activated only after DC power has risen to the regulated voltage levels and must stay low for five milliseconds minimum before the system is allowed to restart. Alternatively, INIT/ may be held low through an open collector device by MPRO/.

How the power failure equipment is configured is left to the system designer. The backup power source may be batteries located on the memory boards or more elaborate facilities located off-board. The location of the power fail logic determines which MULTIBUS power fail lines are used. Pins on the P2 connector have been specified for the power failure functions for use as needed.

To further clarify the location and use of the power fail circuitry, an example of a typical power fail system block diagram is shown in Figure 12. A single board computer and a slave memory board are contained in the system. It is desired to power the memory circuit elements of the memory board from auxiliary power. The single board computer will remain on the main power supply. To accomplish this, user supplied power fail logic and

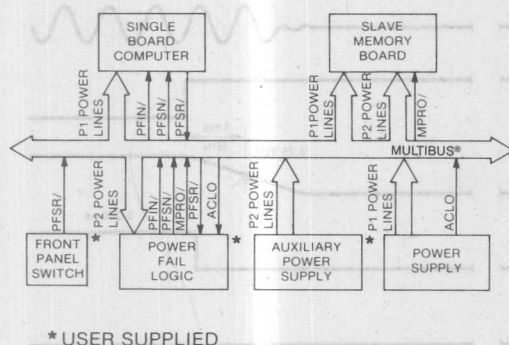


Figure 12. Typical Power Fail System Block Diagram

an auxiliary power supply have been included in the system.

The single board computer is powered from the P1 power lines and accesses the P2 signal lines PFIN/, PFSN/ and PFSR/ (only the P2 signal lines used by a particular functional block are shown on the block diagram). The PFSR/ line is driven from two sources: a front panel switch and the single board computer. The front panel switch is used during normal power-up to reset the power fail sense latch. The single board computer uses the PFSR/ line to reset the latch during a power-up sequence after a power failure. Current single board computers must access the PFSN/ and PFSR/ signals either directly with dedicated circuitry and a P2 pin connection or through the parallel I/O lines with a cable connection from the parallel I/O connector to the P2 connector.

The slave memory board uses both the P1 and P2 power lines, the P2 power lines are used (at all times) to power the memory circuit elements and other support circuits, the P1 power lines power all other circuitry. In addition, the MPRO/ line is input and used to sense when memory contents should be protected.

The power fail logic contains the power fail sense latch, and uses the PFSR/ and ACLO lines for inputs and the PFIN/ PFSN/, and MPRO/ lines for outputs. The power fail logic must be powered by the P2 power lines.

DC Requirements — The drive and load characteristics of the bus signals are listed in Appendix C. The physical locations of the drivers and loads, as well as the terminating resistor value for each bus line, are also specified. Appendix D contains the MULTIBUS power specifications.

MULTIBUS® Slave Interface Circuit Elements

There are three basic elements of a slave bus interface: address decoders, bus drivers, and control signal logic. This section discusses each of these elements in general terms. A description of a detailed implementation of a slave interface is presented in a later section of this application note.

Address Decoding — This logic decodes the appropriate MULTIBUS address bits into RAM requests, ROM requests, or I/O selects. Care must be taken in the design of the address decode logic to ensure flexibility in the selection of base address assignments. Without this flexibility, restrictions may be placed upon various system configurations. Ideally, switches and jumper connections should be associated with the decode logic to permit field modification of base address assignments.

The initial step in designing the address decode portion of a MULTIBUS interface is to determine the required number of unique address locations. This decision is influenced by the fact that address decoding is usually done in two stages. The first stage decodes the base address, producing an enable for the second stage which generates the actual device selects for the user logic. A convenient implementation of this two stage decoding scheme utilizes a pair of decoders driven by the high order bits of the address for the first stage and a second decoder for the low order bits of the address bus. This technique forces the number of unique address locations to be a power of two, based at the address decoded by the first stage. Consider the scheme illustrated in Figure 13.

As shown in Figure 13, the address bits A₄·A₃ are used to produce switch selected outputs of the first stage of decoding. The 1 out of 8 binary decoders

have been used. The top decoder decodes address lines A₄ - A₇, and the bottom decoder decodes address lines A₈ - A_B. If only address lines A₀ - A₇ are being used for device selection, as in the case of I/O port selection in 8-bit systems, the bottom decoder may be disabled by setting switch S₂ to the ground position. Address lines A₇ and A_B drive enable inputs E₂ or E₃ of the decoders. The address lines A₀ - A₃ enter the second stage address decoder to produce 8 user device selects. The second stage decoder must first be enabled by an address that corresponds to the switch-selected base address.

Address decoding must be completed before the arrival of a command. Since the command may become active within 50 ns after stable address, the decode logic should be kept simple with a minimal number of layers of logic. Furthermore, the timing is extremely critical in systems which make use of the inhibit lines.

A linear or unary select scheme in which no binary encoding of device address (e.g., address bit A₀ selects device 0, address bit A₁ selects device 1, etc.) is performed is not recommended because the scheme offers no protection in case multiple

devices are simultaneously selected, and because the addressing within such a system is restricted by the extent of the address space occupied by such a scheme.

Data Bus Drivers — For user designed logic which simply receives data from the MULTIBUS data lines, this portion of the bus interface logic may only consist of buffers. Buffers are required to ensure that maximum allowable bus loading is not exceeded by the user logic.

In systems where the user designed logic must place data onto the MULTIBUS data lines, three-state drivers are required. These drivers should be enabled only when a memory read command (MRDC/) or an I/O read command (IORC/) is present and the module has been addressed.

When both the read and write functions are required, parallel bidirectional bus drivers (e.g., Intel 8226, 8287, etc.) are used. A note of caution must be included for the designer who uses this type of device. A problem may arise if data hold time requirements must be satisfied for user logic following write operations. When bus commands are used to directly produce both the chip select for the bidirectional bus driver and a strobe to a latch in the user logic, removal of that signal may not provide the user's latch with adequate data hold time. Depending on the specifics of the user logic, this problem may be solved by permanently enabling the data buffer's receiver circuits and controlling only the direction of the buffers.

Control Signal Logic — The control signal logic consists of the circuits that forward the I/O and memory read/write commands to their respective destinations, provide the bus with a transfer acknowledge response, and drive the system interrupt lines.

Bus Command Lines

The MULTIBUS information transfer protocol lines (MRDC/, MWTC/, IORD/, and IOWC/) should be buffered by devices with very high speed switching. Because the bus DC requirements specify that each board may load these lines with 2.0 mA, Schottky devices are recommended. LS devices are not recommended due to their poor noise immunity. The commands should be gated

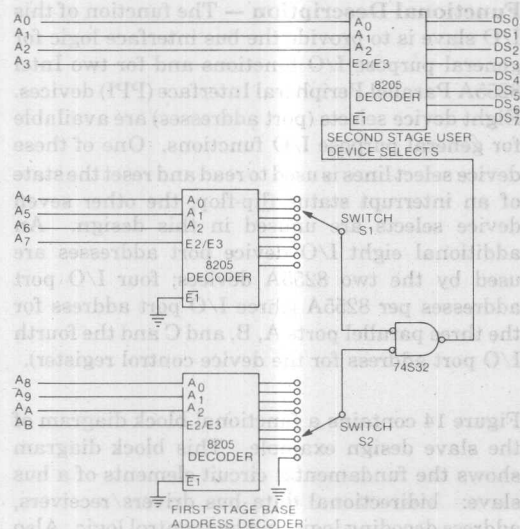


Figure 13. Two Stage Decoding Scheme

with a signal indicating the base address has been decoded to generate read and write strobes for the user logic.

Transfer Acknowledge Generation

The user interface transfer acknowledge generation logic provides a transfer acknowledge response, XACK/, to notify the bus master that write data provided by the bus master has been accepted or that read data it has requested is available on the MULTIBUS data lines. XACK/ allows the bus master to conclude its current instruction.

Since XACK/ timing requirements depend on both the CPU of the bus master and characteristics of the user logic, a circuit is needed which will provide a range of easily modified acknowledge responses.

The transfer acknowledge signals must be driven by three-state drivers which are enabled when the bus interface is addressed and a command is present.

Interrupt Signal Lines

The asynchronous interrupt lines must be driven by open collector devices with a minimum drive of 16 mA.

In a typical Non Bus Vectored Interrupt system, logic must be provided to assert and latch-up an interrupt signal. In addition to driving the MULTIBUS interrupt lines, the latched interrupt signal would be read by an I/O operation such as reading the module's status. The interrupt signal would be cleared by writing to the status register.

III. MULTIBUS® SLAVE DESIGN EXAMPLE

A MULTIBUS slave design example has been included in this application note to reinforce the theory previously discussed. The design example is of general purpose I/O slave interface. This design example could easily be modified to be used as a slave memory interface by buffering the address signals and using the appropriate MULTIBUS memory commands. In addition, to help the reader better understand an application for an I/O slave interface, two Intel 8255A Parallel Peripheral Interface (PPI) devices are shown connected to the slave interface.

The design example is shown in both 8/16-bit version and an 8-bit version. The 8/16-bit version

is an I/O interface which will permit a 16-bit master to perform 8 or 16 bit data transfers. 8-bit masters may also use the 8/16-bit version of the design example to perform 8-bit data transfers.

The 8-bit version of the design example may be used by both 8 or 16-bit masters, but will only perform 8-bit data transfers. It does not contain the circuitry required to perform 16-bit data transfers.

Both the 8/16-bit version and the 8-bit version of the design example were implemented on an iSBC 905 prototype board. The schematics for each of the examples are given in Appendices F and G.

Functional/Programming Characteristics

This section describes the organization of the slave interface from two points of view, the functional point of view and the programming characteristics. First, the principal functions performed by the hardware are identified and the general data flow is illustrated. This point of view is intended as an introduction to the detailed description provided in the next section; Theory of Operation. In the second point of view, the information needed by a programmer to access the slave is summarized.

Functional Description — The function of this I/O slave is to provide the bus interface logic for general purpose I/O functions and for two Intel 8255A Parallel Peripheral Interface (PPI) devices. Eight device selects (port addresses) are available for general purpose I/O functions. One of these device select lines is used to read and reset the state of an interrupt status flip-flop, the other seven device selects are unused in this design. An additional eight I/O device port addresses are used by the two 8255A devices; four I/O port addresses per 8255A (three I/O port address for the three parallel ports A, B, and C and the fourth I/O port address for the device control register).

Figure 14 contains a functional block diagram of the slave design example. This block diagram shows the fundamental circuit elements of a bus slave: bidirectional data bus drivers/receivers, address decoding logic and bus control logic. Also shown is the address decoding logic for the low order four bits, the interrupt logic which is selected by this decoding logic, and the two 8255A devices.

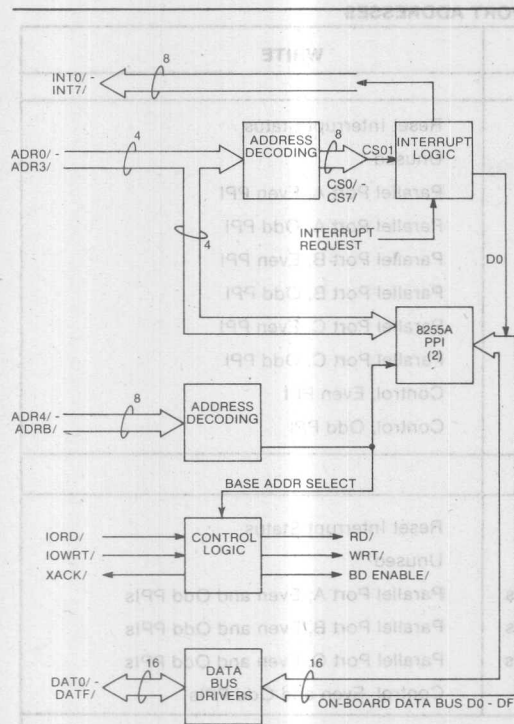


Figure 14. MULTIBUS® Slave Design Example Functional Block Diagram

Programming Characteristics — The slave design example provides 16 I/O port addresses which may be accessed by user software. The base address of the 16 contiguous port addresses is selected by wire wrap connections on the prototype board. The wire wrap connections specify address bits ADR4/- - ADRB/-. They allow the selection of a base address on any 16 byte boundary. Twelve address bits (ADR0/- - ADRB/-) are used since 16-bit (8086 based) masters use 12 bits to specify I/O port addresses. If an 8 bit (8080 or 8085 based) master is used with this slave board, the high order address bits (ADR8/- - ADRB/-) must not be used by the decoding circuits; a wire wrap jumper position (ground position) is provided for this.

The 16 I/O port addresses are divided into two groups of 8 port addresses by decoding address line ADR3/-. Port addresses XX0 - XX7 are used for general I/O functions (XX indicates any hexadecimal digit combination). Port address XX0 is used for accessing the interrupt status flip-flop and

port addresses XX1 - XX7 are not used in this example. Port addresses XX8 - XXF are used for accessing the PPIs. If port addresses XX8 - XXF are selected, then ADR0/- is used to specify which of two PPIs are selected. If the address is even (XX8, XXA, XXC, or XXE) then one PPI is selected. If the address is odd (XX9, XXB, XXD, or XXF), then the other PPI is selected. ADR1/- and ADR2/- are connected directly to the PPIs. Table 1 summarizes the I/O port addresses of the slave design example. Note that if a 16-bit master is used, it is possible to access the slave in a byte or word mode. If word access is used with port address XX8, XXA, XXC, or XXE, then 16 bit transfers will occur between the PPIs and the master. These 16 bit transfers occur because an even address has been specified and the MULTIBUS BHEN/- signal indicates that a 16-bit transfer is requested.

Theory of Operation

In the preceding section, each of the slave design example functional blocks was identified and briefly explained. This section explains how these functions are implemented. For detailed circuit information, refer to the schematics in Appendices F and G. The schematic in Appendix F is on a foldout page so that the following text may easily be related to the schematic.

The discussion of the theory of operation is divided into five segments, each of which discusses a different function performed by the MULTIBUS slave design example. The five segments are:

1. Bus address decoding
2. Data buffers
3. Control signals
4. Interrupt logic
5. PPI operation

Each of these topics are discussed with regard to the 8/16-bit version of the design example; followed by a discussion of the circuit elements which are required by the 8-bit version of the interface.

Bus Address Decoding — Bus address decoding is performed by two 8205 1 out of 8 binary decoders. One decoder (A3) decodes address bits ADR8/- - ADRB/- and the second decoder (A2) decodes address bits ADR4/- - ADR7/-. The base address

Table 1
SLAVE DESIGN EXAMPLE PORT ADDRESSES

I/O PORT ADDRESS	READ	WRITE
BYTE ACCESS		
XX0	Bit 0 = Interrupt Status	Reset Interrupt Status
XX1 - XX7	Unused	Unused
XX8	Parallel Port A, Even PPI	Parallel Port A, Even PPI
XX9	Parallel Port A, Odd PPI	Parallel Port A, Odd PPI
XXA	Parallel Port B, Even PPI	Parallel Port B, Even PPI
XXB	Parallel Port B, Odd PPI	Parallel Port B, Odd PPI
XXC	Parallel Port C, Even PPI	Parallel Port C, Even PPI
XXD	Parallel Port C, Odd PPI	Parallel Port C, Odd PPI
XXE	Illegal Condition	Control, Even PPI
XXF	Illegal Condition	Control, Odd PPI
WORD ACCESS		
XX0	Bit 0 = Interrupt Status	Reset Interrupt Status
XX2 - XX6	Unused	Unused
XX8	Parallel Port A, Even and Odd PPIs	Parallel Port A, Even and Odd PPIs
XXA	Parallel Port B, Even and Odd PPIs	Parallel Port B, Even and Odd PPIs
XXC	Parallel Port C, Even and Odd PPIs	Parallel Port C, Even and Odd PPIs
XXE	Illegal Condition	Control, Even and Odd PPIs

XX = Any hex digits, assigned by jumpers; XX defines the base address.

selected is determined by the position of wire wrap jumpers. The outputs of the two decoders are ANDed together to form the BASE ADRSELECT/ signal. This signal specifies the base address for a group of 16 I/O ports. Using the wire wrap jumper positions shown in the schematic, a base address of E3 has been selected. Therefore, this MULTIBUS slave board will respond to I/O port addresses in the E30 - E3F range.

If this slave board is to be used with 8-bit MULTIBUS masters, the high order address bits must not be decoded. Therefore, the wire wrap jumper which selects the output of decoder A3 must be placed in the top (ground) position (pin 10 of gate A9 to ground).

The low order 4 address lines (ADR0/- ADR3/) are buffered and inverted using 74LS04 inverters. These address lines are input to an 8205 for decoding a chip select for the interrupt logic; the address lines are also used directly by the PPIs. LS-Series logic is required for buffering to meet the MULTIBUS specification for I_{IL} (low level input

current). S-Series or standard series logic will not meet this specification.

Address decoder A4 is used to decode addresses E30 - E37. The CS0/ output of this decoder is used to select the interrupt logic, thus I/O port address E30 is used to read and reset the interrupt latch. The remaining outputs from decoder A4 (CS1/- CS7/) are not used in this example. They would normally be used to select other functions in a slave board with more capability. Note that in the schematic shown in Appendix G for the 8-bit version of this slave design example, the high order (ADR8/- ADRB/) address decoder is not included and the BHEN/ signal is not used.

Data Buffers — Intel 8287 8-bit parallel bi-directional bus drivers are used for the MULTIBUS data lines DAT0/- DATF/. In the 8/16-bit version of the slave board, three 8287 drivers are used.

When an 8-bit data transfer is requested, either driver A5, which is connected to on-board data

lines D0 - D7, or driver A6, which is connected to on-board data lines D8 - DF, is used. If a byte transfer is requested from an even address, driver A5 will be selected. If a byte transfer from an odd address is requested, driver A6 will be selected. All byte transfers take place on MULTIBUS data lines DAT0/ - DAT7/. When a word (16-bit) transfer is requested from an even address, drivers A5 and A7 will be used. Note that if a user program requests a word transfer from an odd address, 16-bit masters in the iSBC product line will actually perform two byte transfer requests.

The logic which determines the chip selection (8287 input signal OE, output enable) signals for the bus drivers uses the low order address bit (ADR0/) and the buffered Byte High Enable signal (BHENBL/). Note that the MULTIBUS signal BHEN/ has been buffered with an 74LS04 inverter. This is done to meet the bus address line loading specification. The SWAP BYTE/ signal which is generated is qualified by the BD ENBL/ signal and used to select the bus drivers.

The steering pin for the 8287 drivers is labelled T (transmit) and is driven by the signal RD. When an input (read) request is active or when neither a read or write command is being serviced, the direction of data transfer of the 8287 will be set for B to A.

The 8287 drivers are set to point IN (direction B to A) when no MULTIBUS I/O transfer command is being serviced for two reasons. First, if the driver were pointed OUT (direction A to B) and a write command occurred, it would be necessary to turn the buffers IN and set the OE (output enable) signal active before the data could be transferred to the on-board bus. A possibility of a "buffer-fight" could occur in some designs if the OE signal permitted an 8287 to drive the MULTIBUS data lines momentarily before the steering signal could switch the direction of the 8287. In this case, both the MULTIBUS master and the slave would be driving the data lines; this is not recommended. (In this particular design, the steering signal will always stabilize before the OE signal becomes active.)

The second reason the driver is pointing IN when no command is present is due to the "data valid after WRITE" requirements of the 8255As. The 8255A requires that data remain on its data lines for 30 ns after the WRITE command (WR at the 8255A) is removed. This requirement will be met if the direction of the 8287 drivers is not switched

when the MULTIBUS IOWC/ signal is removed (WRT/ could have been used to steer the 8287 instead of RD); and if the capacitance of the on-board data bus lines is sufficient to hold the data values on the bus after the 8287 OE signal and the 8255A PPI WRT/ signal go inactive. The on-board data bus may easily be designed such that the capacitance of the lines is sufficient to meet the 30 ns data hold time requirement. In addition, the current leakage of all devices connected to the on-board bus must be kept small to meet the 30 ns data hold time requirement.

The 8-bit version of this design example uses only one 8287 instead of the three required by the 8/16-bit version. The logic required to control the swap byte buffer is also not necessary. The chip select signal used for the 8287 is the BD ENBL/ signal.

Control Signals — The MULTIBUS control signals used by this slave design example are IORC/, IOWC/, and XACK/. IORC/ and IOWC/ are qualified by the BASE ADR SELECT/ signal to form the signals RD and WRT. RD and WRT are used to drive the interrupt logic, the PPI logic and the XACK/ (transfer acknowledge) logic.

For the XACK/ logic RD and WRT are ORed to form the BD ENBL/ signal which is inverted and used to drive the CLEAR pin of a shift register. When the slave board is not being accessed, the CLEAR pin of the shift register will be low (BD ENBL/ is high). This causes the shift register to remain cleared and all outputs of the shift register will be low. When the slave board is accessed, the CLEAR pin will be high, and the A and B inputs (which are high) will be clocked to the output pins by CCLK/. To select a delay for the XACK/ signal, a jumper must be installed from one of the shift register output pins to the 8089 tri-state driver. Each of the shift register output pins select an integer multiple of CCLK/ periods for the signal delay. Since the CCLK/ signal is asynchronous, the actual delay selected may only be specified with a tolerance of one CCLK/ period. In this example a delay of 3 - 4 CCLK/ periods was selected; with a CCLK/ period of 100 ns, the XACK/ delay would occur somewhere within the range of 300 - 400 ns from the time when the CLEAR signal goes high.

The control signal logic used in the 8-bit version of the slave design example is identical to the logic used in the 8/16-bit version.

Interrupt Logic — The interrupt logic uses a 74S74 flip-flop to latch an asynchronous interrupt request from some external logic. The Q output of the INTERRUPT REQUEST LATCH is output through an open collector gate to one of the MULTIBUS interrupt lines. The state of the INTERRUPT REQUEST LATCH is transferred to the INTERRUPT STATUS LATCH when a read command is performed on I/O port BASE ADDRESS+0 (E30 for the jumper configuration shown). The Q output of INTERRUPT STATUS LATCH is used to drive data line D0 of the on-board data bus by using an 8089 tri-state driver. If a user program performs an INPUT from I/O port E30, data bit 0 will be set to 1 if the INTERRUPT REQUEST LATCH is set.

The purpose of INTERRUPT STATUS LATCH is to minimize the possibility of the asynchronous interrupt occurring while the interrupt status is being read by a bus master. If the latch was not included in the design and an asynchronous interrupt did occur while a bus master is reading MULTIBUS data line DAT0/, a data buffer on the master could go into a meta-stable state. By adding the extra latch, which is clocked by the IORD/ command for I/O port E30, the possibility of data line DAT0/ changing during a bus master read operation is eliminated.

The INTERRUPT REQUEST LATCH is cleared when a user program performs an OUTPUT to I/O port E30.

This interrupt structure assumes that several interrupt sources may exist on the same MULTIBUS interrupt line (for example, INT3/). When the MULTIBUS master gets interrupted, it must poll the possible sources of the interrupt received and after determining the source of the interrupt, it must clear the INTERRUPT REQUEST LATCH for that particular interrupt source.

The interrupt logic for the 8-bit version of the design example is identical to the interrupt logic of the 8/16-bit version of the design example.

PPI Operation — Two 8255A Parallel Peripheral Interface (PPI) devices are shown interfaced to the slave design example logic. One PPI is connected to the on-board data bus lines D0 - D7 and is addressed with the even I/O port addresses E38, E3A, E3C, and E3E. The second PPI is connected to data bus lines D8 - DF and is addressed with the odd I/O port addresses E39, E3B,

E3D, and E3F. The even or odd I/O port selection is controlled by using the ADR0 address line in the chip select term of the PPIs. In addition, the odd PPI (A11) is selected when the BHENBL term is high. This occurs when the MULTIBUS signal BHEN/ is low indicating that a word (16-bit) I/O instruction is being executed. When a word I/O instruction is executed, both PPIs will perform the I/O operation specified.

The specifications of the 8255A device state that the address lines A0 and A1 and the chip select lines must be stable before the \overline{RD} or \overline{WR} lines are activated. The MULTIBUS specification address set-up time of 50 ns and the short gate propagation delays in this design assure that the address lines are stable before \overline{RD} or \overline{WR} are active.

The data hold requirements of the 8255A were discussed in a previous section. The 8255A specification states that data will be stable on the data bus lines a maximum of 250 ns after a READ command. This specification was used to select the delay for the XACK/ signal.

The PPI operation for the 8-bit version of the design example is slightly different than that used for the 8/16-bit version. The chip select signal for the bottom PPI does not use the BHENBL term since 16-bit data transfers are not possible with an 8-bit I/O slave board. Also, the chip select and address signals have been swapped so the top PPI occupies I/O address range X8 - XB, and the bottom PPI occupies I/O address range XC - XF (X is the base address of the 8-bit version). This swapping of the address lines was not necessary; however, it was thought to be more convenient to access the PPIs in two groups of 4 contiguous I/O port addresses.

IV. SUMMARY

This application note has shown the structure of the Intel MULTIBUS system bus. The structure supports a wide range of system modules from the Intel OEM Microcomputer Systems product line that can be extended with the addition of user designed modules. Because the user designed modules are no doubt unique to particular applications, a goal of this application note has been to describe in detail the singular common element - the bus interface. Material has also been presented to assist the systems designer to understanding the bus functions so that successful systems integration can be achieved.

Appendix

Contents

DESCRIPTION	MEMORIC	PIN	CONTENTS	PAGE
Signal GND	GND	2	APPENDIX A — MULTIBUS® PIN ASSIGNMENTS	22
+5Vdc	+5V	4	APPENDIX B — BUS TIMING SPECIFICATIONS	24
+3Vdc	+3V	6	APPENDIX C — BUS DRIVERS, RECEIVERS, AND TERMINATIONS	26
+15Vdc	+15V	8	APPENDIX D — BUS POWER SUPPLY SPECIFICATIONS	28
-5Vdc	-5V	10	APPENDIX E — MECHANICAL SPECIFICATIONS	29
Signal GND	GND	12	APPENDIX F — MULTIBUS® SLAVE DESIGN EXAMPLE SCHEMATIC 8/16-BIT VERSION	31
INIT	INIT	24	APPENDIX G — MULTIBUS® SLAVE DESIGN EXAMPLE SCHEMATIC 8-BIT VERSION	33
Bus Pk. Out	BPRQ	26		
Bus Request	BREQ	28		
Mem Write Cmd	WTC	30		
IO Write Cmd	WOC	32		
Mem Read Cmd	MRQ	34		
Address	ADR	36		
Bus	BUS	38		
Parallel Interrupt Request	INT	40		
Serial Interrupt Request	INT	42		
Address	ADR	44		
Bus	BUS	46		
Address	ADR	48		
Bus	BUS	50		
Address	ADR	52		
Bus	BUS	54		
Address	ADR	56		
Bus	BUS	58		
DATA	DATA	60		
DATA	DATA	62		
DATA	DATA	64		
DATA	DATA	66		
DATA	DATA	68		
DATA	DATA	70		
DATA	DATA	72		
DATA	DATA	74		
Signal GND	GND	76		
Reserved	RES	78		
+15Vdc	+15V	80		
+3Vdc	+3V	82		
+5Vdc	+5V	84		
Signal GND	GND	86		

APPENDIX A

PIN ASSIGNMENT OF BUS SIGNALS ON MULTIBUS® BOARD P1 CONNECTOR

	PIN	(COMPONENT SIDE)		PIN	(CIRCUIT SIDE)	
		MNEMONIC	DESCRIPTION		MNEMONIC	DESCRIPTION
POWER SUPPLIES	1	GND	Signal GND	2	GND	Signal GND
	3	+5V	+5Vdc	4	+5V	+5Vdc
	5	+5V	+5Vdc	6	+5V	+5Vdc
	7	+12V	+12Vdc	8	+12V	+12Vdc
	9	-5V	-5Vdc	10	-5V	-5Vdc
	11	GND	Signal GND	12	GND	Signal GND
BUS CONTROLS	13	BCLK/	Bus Clock	14	INIT/	Initialize
	15	BPRN/	Bus Pri. In	16	BPRO/	Bus Pri. Out
	17	BUSY/	Bus Busy	18	BREQ/	Bus Request
	19	MRDC/	Mem Read Cmd	20	MWTC/	Mem Write Cmd
	21	IORC/	I/O Read Cmd	22	IOWC/	I/O Write Cmd
	23	XACK/	XFER Acknowledge	24	INH1/	Inhibit 1 disable RAM
BUS CONTROLS AND ADDRESS	25		Reserved	26	INH2/	Inhibit 2 disable PROM or ROM
	27	BHEN/	Byte High Enable	28	AD10/	Address Bus
	29	CBRQ/	Common Bus Request	30	AD11/	
	31	CCLK/	Constant Clk	32	AD12/	
	33	INTA/	Intr Acknowledge	34	AD13/	
INTERRUPTS	35	INT6/	Parallel Interrupt Requests	36	INT7/	Parallel Interrupt Requests
	37	INT4/		38	INT5/	
	39	INT2/		40	INT3/	
	41	INT0/		42	INT1/	
ADDRESS	43	ADRE/	Address Bus	44	ADRF/	Address Bus
	45	ADRC/		46	ADRD/	
	47	ADRA/		48	ADRB/	
	49	ADR8/		50	ADR9/	
	51	ADR6/		52	ADR7/	
	53	ADR4/		54	ADR5/	
	55	ADR2/		56	ADR3/	
	57	ADR0/		58	ADR1/	
DATA	59	DATE/	Data Bus	60	DATF/	Data Bus
	61	DATC/		62	DATD/	
	63	DATA/		64	DATB/	
	65	DAT8/		66	DAT9/	
	67	DAT6/		68	DAT7/	
	69	DAT4/		70	DAT5/	
	71	DAT2/		72	DAT3/	
	73	DAT0/		74	DAT1/	
POWER SUPPLIES	75	GND	Signal GND	76	GND	Signal GND
	77		Reserved	78		Reserved
	79	-12V	-12Vdc	80	-12V	-12Vdc
	81	+5V	+5Vdc	82	+5V	+5Vdc
	83	+5V	+5Vdc	84	+5V	+5Vdc
	85	GND	Signal GND	86	GND	Signal GND

All Mnemonics © Intel Corporation 1978

APPENDIX A (Continued)

P2 CONNECTOR PIN ASSIGNMENT OF OPTIONAL BUS SIGNALS

(COMPONENT SIDE)			(CIRCUIT SIDE)		
PIN	MNEMONIC	DESCRIPTION	PIN	MNEMONIC	DESCRIPTION
1	GND	Signal GND	2	GND	Signal GND
3	5 VB	+5V Battery	4	5 VB	+5V Battery
5		Reserved	6	VCCPP	+5V Pulsed Power
7	-5 VB	-5V Battery	8	-5 VB	-5V Battery
9		Reserved	10	Reserved	
11	12 VB	+12V Battery	12	12 VB	+12V Battery
13	PFSR/	Power Fail Sense Reset	14	Reserved	
15	-12 VB	-12V Battery	16	-12 VB	-12V Battery
17	PFSN/	Power Fail Sense	18	ACLO	AC Low
19	PFIN/	Power Fail Interrupt	20	MPRO/	Memory Protect
21	GND	Signal GND	22	GND	Signal GND
23	+15V	+15V	24	+15V	+15V
25	-15V	-15V	26	-15V	-15V
27	PAR1/	Parity 1	28	HALT/	Bus Master HALT
29	PAR2/	Parity 2	30	WAIT/	Bus Master WAIT STATE
31			32	ALE	Bus Master ALE
33			34	Reserved	
35			36	Reserved	
37			38	AUX RESET/	Reset switch
39			40		
40			42		
43			44		
45			46		
47			48		
49			50		
51			52		
53			54		
55			56		
57			58		
59			60		
Notes:					
1. PFIN, on slave modules, if possible, should have the option of connecting to INT0/ on P1.					
2. All undefined pins are reserved for future use.					
All Mnemonics © Intel Corporation 1978					

APPENDIX B
BUS TIMING SPECIFICATIONS SUMMARY

Parameter	Description	Minimum	Maximum	Units
t _{BCY}	Bus Clock Period	100	D.C. 0.5V Battery	ns
t _{BW}	Bus Clock Width	0.35 t _{BCY}	0.65 t _{BCY}	
t _{SKEW}	BCLK/skew		(Not Restricted)	
t _{PD}	Standard Bus Propagation Delay		3	ns
t _{AS}	Address Set-Up Time (at Slave Board)	50		ns
t _{DS}	Write Data Set Up Time	50		ns
t _{AH}	Address Hold Time	50		ns
t _{DHW}	Write Data Hold Time	50		ns
t _{DXL}	Read Data Set Up Time To XACK	0		ns
t _{DHR}	Read Data Hold Time	0	65	ns
t _{XAH}	Acknowledge Hold Time	0	65	ns
t _{XACK}	Acknowledge Time	0	8	μs
t _{CMD}	Command Pulse Width	100	9.5	ns
t _{ID}	Inhibit Delay	0	100 (Recommend < 100 ns)	ns
t _{XACKA}	Acknowledge Time of an Inhibited Slave	t _{IAD} + 50 ns	1500	
t _{XACKB}	Acknowledge Time of an Inhibiting Slave	1.5	8	μs
t _{IAD}	Acknowledge Disable from Inhibit (An internal parameter on an inhibited slave; used to determine t _{XACKA} Min.)	0	100 (arbitrary)	ns
t _{AIZ}	Address to Inhibits High Delay		1.0	ns
t _{INTA}	INTA / Width	250		ns
t _{CSEP}	Command Separation	100		ns

AP-28A

APPENDIX B (Continued)
BUS TIMING SPECIFICATIONS SUMMARY

Parameter	Description	Minimum	Maximum	Units
tBREQL	↓BCLK/ to BREQ/ Low Delay	0	35	ns
tBREQH	↓BCLK/ to BREQ/ High Delay	0	35	ns
tBPRNS	BPRN/ to ↓BCLK/ Setup Time	22		ns
tBUSY	BUSY/ delay from ↓BCLK/	0	70	ns
tBUSYS	BUSY/ to ↓BCLK/ Setup Time	25		ns
tBPRO	↓BCLK/ to BPRO/ (CLK to Priority Out)	0	40	ns
tBPRNO	BPRN/ to BPRO/ (Priority In to Out)	0	30	ns
tCBRO	↓BCLK/ to CBRQ/ (CLK to Common Bus Request)	0	60	ns
tCBROS	CBRQ/ to ↓BCLK/ Setup Time	35		ns
tXCD	XACK/ to Command/ Delay		1500	ns
tBSYO	CBRQ/I and BUSY/I to BUSY/I		12	μs
tCCY	C-clock Period	100	110	ns
tCW	C-clock Width	0.35 tCCY	0.65 tCCY	ns
tINIT	INIT/ Width	5		ms
tINITS	INIT/ to MPRO/ Setup Time	100		ns
tPBD	Power Backup Logic Delay	0	200	ns
tFINW	PFIN/ Width	2.5		ms
tMPRO	MPRO/ Delay	2.0	2.5	ms
tACLOW	ACLO/ Width	3.0		ms
tPFSRW	PFSR/ Width	100		ns
tTOUT	Timeout Delay	5	∞ (D.C.)	ms
tDCH	D.C. Power Supply Hold from ALCO/	3.0		ms
tDCS	D.C. Power Supply Setup to ACLO/	5		ms

APPENDIX C
BUS DRIVERS, RECEIVERS, AND TERMINATIONS

Bus Signals	Driver 1,3					Receiver 2,3				Termination		
	Location	Type	IOL	IOH	CO	Location	IIL	IiH	Ci	Location	Type	R Units
			Min _{ma}	Min _{μa}	Max _{pl}		Max _{ma}	Max _{μa}	Max _{pl}			
DAT0/→DATF/ (16 lines)	Masters and Slaves	TRI	16	-2000	300	Masters and Slaves	-0.8	125	18	1 place	Pullup	2.2 KΩ
ADR0/→ADRB/	Masters	TRI	16	-2000	300	Slaves	-0.8	125	18	1 place	Pullup	2.2 KΩ
BHEN/ (21 lines)												
MRDC/, MWTC/	Masters	TRI	32	-2000	300	Slaves (Memory; memory- mapped I/O)	-2	125	18	1 place	Pullup	1 KΩ
IORC/, IOWC/	Masters	TRI	32	-2000	300	Slaves (I/O)	-2	125	18	1 place	Pullup	1 KΩ
XACK/	Slaves	TRI	32	-2000	300	Masters	-2	125	18	1 place	Pullup	510 Ω
INH1/, INH2/	Inhibiting Slaves	OC	16	—	300	Inhibited Slaves (RAM, PROM, ROM, Memory- Mapped I/O)	-2	50	18	1 place	Pullup	1 KΩ
BCLK/	1 place (Master us)	TTL	48	-3000	300	Master	-2	125	18	Mother- board	To +5V To GND	220 Ω 330 Ω
BREQ/	Each Master	TTL	5	-200	60	Central Priority Module	2	50	18	Central Priority Module (not req)	Pullup	1 KΩ
BPRO/	Each Master	TTL	5	-200	60	Next Master in Serial Priority Chain at its BPRN/	-1.6	50	18	(not req)		
BPRN/	Parallel: Central Priority Module Serial: Prev Masters BPRO/	TTL	5	-200	300	Master	-4	100		(not req)		
BUSY/, CBRQ	All Masters	O.C.	20	—	300	All Masters	-2	50	18	1 place	Pullup	1 KΩ
INIT/	Master	O.C.	32	—	300	All	-2	50	18	1 place	Pullup	2.2 KΩ
CCLK/	1 place	TTL	48	-3000	300	Any	-2	125	18	Mother- board	To +5V To GND	220 Ω 330 Ω
INTA/	Masters	TRI	32	-2000	300	Slaves (Interrupting I/O)	-2	125	18	1 place	Pullup	1 KΩ
INT0/→INT7/ (8 lines)	Slaves	O.C.	16	—	300	Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFSR/	User's Fron Panel?	TTL	16	-400	300	Slaves, Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFSN/	Power Back Up Unit	TTL	16	-400	300	Masters	-1.6	40	18	1 place	Pullup	1 KΩ
ACLO	Power Supply	O.C.	16	-400	300	Slaves, Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFIN/	Power Back- Up Unit	O.C.	16	-400	300	Masters	-1.6	40	18	1 place	Pullup	1 KΩ
MPRO/	Power Back- Up Unit	TTL	16	-400	300	Slaves Masters	-1.6	40	18	1 place	Pullup	1 KΩ

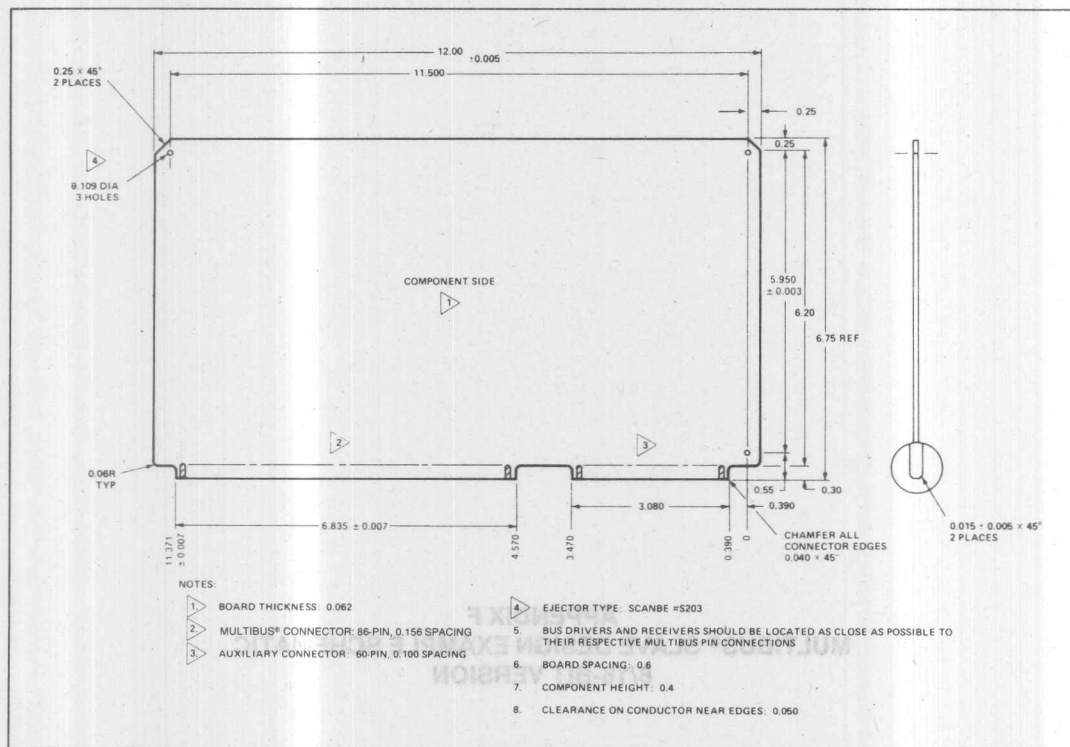
APPENDIX C (Continued)
BUS DRIVERS, RECEIVERS, AND TERMINATIONS

Driver 1,3				Receiver 2,3				Termination		
Bus Signals	Location	Type	I _{OL} Min _{ma} I _{OH} Min _{μa} C _O Max _{pf}	Location	I _{IL} Max _{ma} I _{IH} Max _{μa} C _I Max _{pf}			Location	Type	R Units
Aux Reset/	User's Front Panel?	Switch to GND (Note 5)	— — —	Masters	2 50 18			None		
Notes: 1. Driver Requirements I _{OH} = High Output Current Drive I _{OL} = Low Output Current Drive C _O = Capacitance Drive Capability TRI = 3-State Drive O.C. = Open Collector Driver TTL = Totem-pole Driver 2. Receiver Requirements I _{IH} = High Input Current Load I _{IL} = Low Input Current Load C _I = Capacitive Load 3. TTL low state must be $\geq -0.5v$ but $\leq 0.8v$ at the receivers TTL high state must be $\geq 2.0v$ but $\leq 5.5v$ at the receivers 4. For the iSBC 80/10 and the iSBC 80/10A use only a 1K pull-up resistor to +5v for BCLK/ and CCLK/ termination. 5. Recommend a 47Ω resistor in series with switch.										

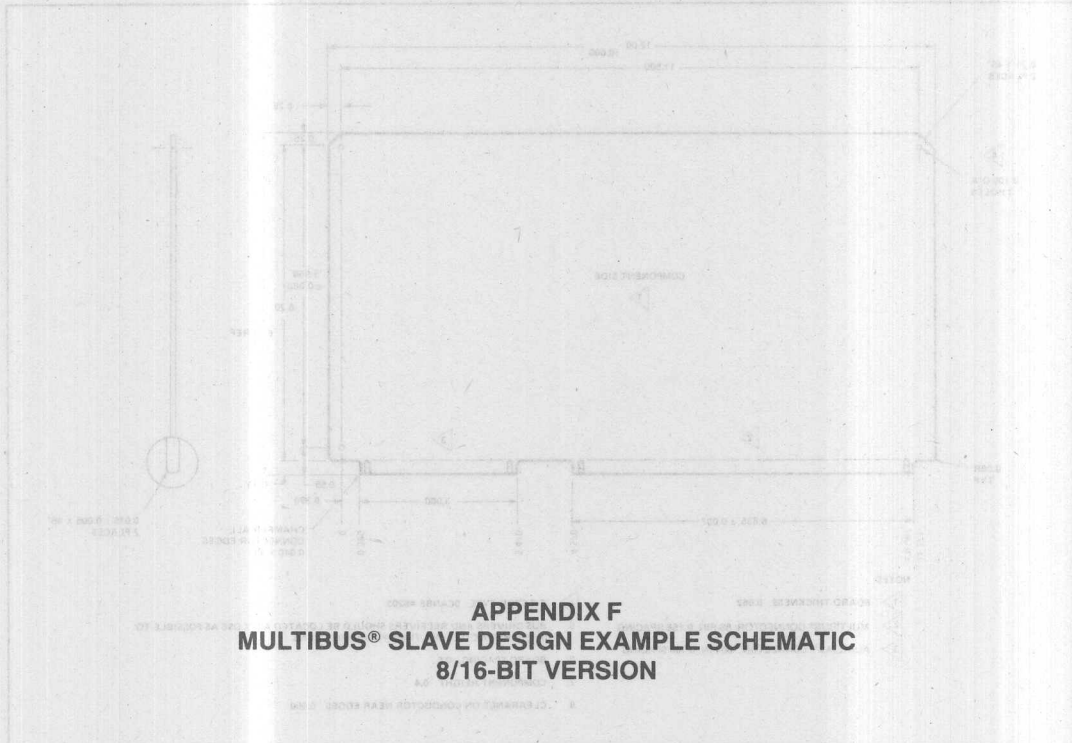
APPENDIX D
BUS POWER SPECIFICATIONS

	Standard (P1)				Optional (P2)					
	Ground				Analog Power		Battery Power Backup			
		+ 5	+ 12	- 12	+ 15	- 15	+ 5	+ 12	- 12	- 5
Mnemonic	GND	+ 5V	+ 12V	- 12V	+ 15V	- 15V	+ 5B	+ 12B	- 12B	- 5B
Bus Pins	P1 + 1,2, 11,12, 75,76 85,86	P1 + 3,4, 5,6,81, 82,83, 84	P1 + 7,8	P1 + 79, 80	P2 + 23, 24	P2 + 25, 26	P2 + 3,4, 5,6	P2 + 11, 12	P2 + 15, 16	P2 - 7,8
Nominal Output	Ref.	+ 5.0V	+ 12.0V	- 12.0V	+ 15.0V	- 15.0V	+ 5.0V	+ 12.0V	- 12.0V	- 5.0V
Tolerance from Nominal ¹	Ref.	± 5%	± 5%	± 5%	± 3%	± 3%	± 5%	± 5%	± 5%	± 5%
Ripple (Pk-Pk) ²	Ref.	50 mV	50 mV	50 mV	10 mV	10 mV	50 mV	50 mV	50 mV	50 mV
Transient Response Time ³		500 μs	500 μs	500 μs	100 μs	100 μs	500 μs	500 μs	500 μs	500 μs
Transient Deviation ⁴		± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%
NOTES: 1. Tolerance is worst case, including initial voltage setting line and load effects of power source, temperature drift, and any additional steady state influences. 2. As measured over any bandwidth not to exceed 0 to 500 kHz. 3. As measured from the start of a load change to the time an output recovers within ± 0.1% of final voltage. 4. Measured as the peak deviation from the initial voltage.										

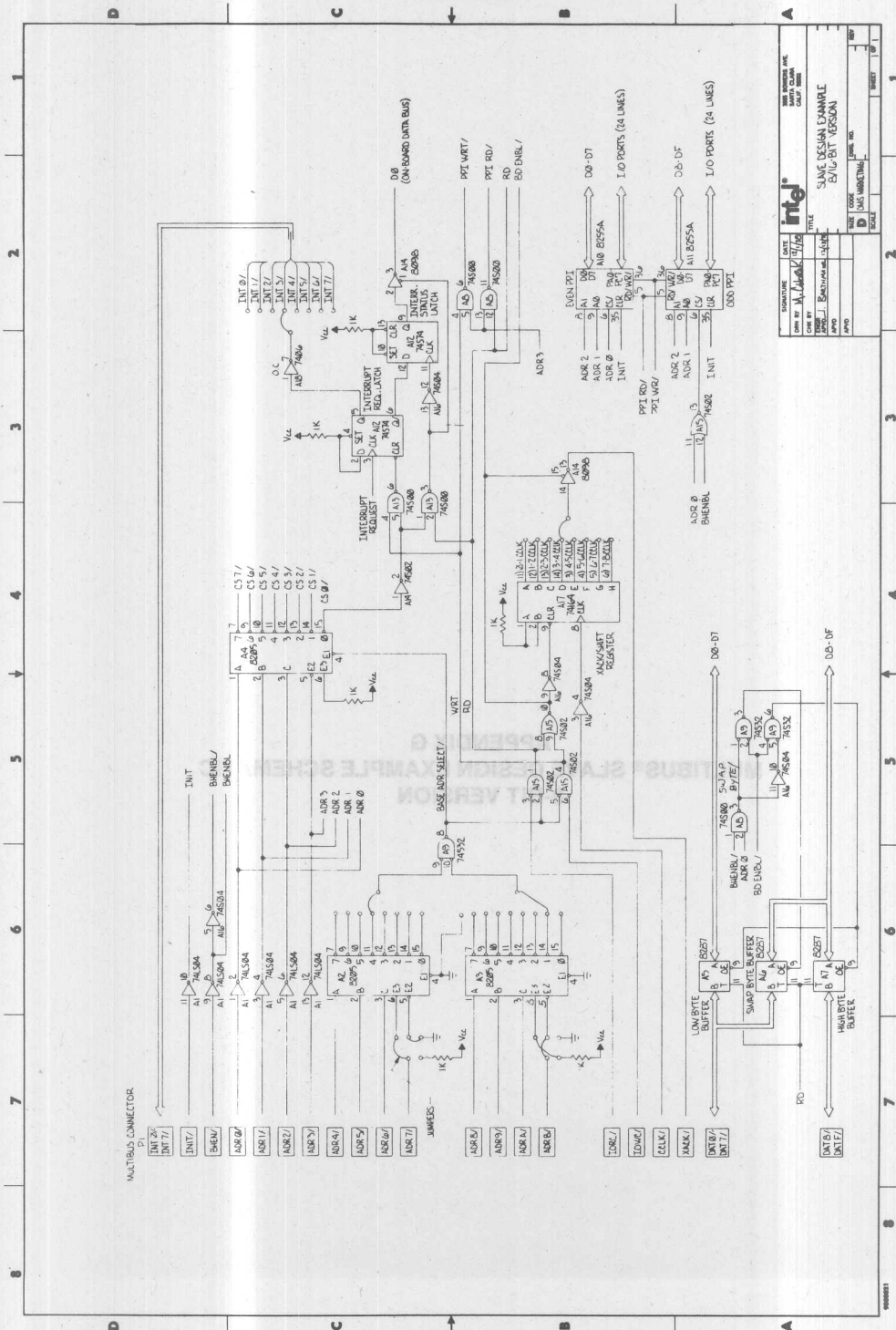
APPENDIX E MECHANICAL SPECIFICATIONS



MECHANICAL SPECIFICATIONS
APPENDIX E



AP-28A
APPENDIX F



MULTIBUS® SLAVE DESIGN EXAMPLE SCHEMATIC 8/16-BIT VERSION

APPENDIX B
BUS TIMING SPECIFICATIONS SUMMARY

Parameter	Description	Minimum	Maximum	Units
t_{BCY}	Bus Clock Period	100	D.C.	ns
t_{BW}	Bus Clock Width	$0.35 t_{BCY}$	$0.65 t_{BCY}$ (Not Restricted)	
t_{SKEW}	BCLK/skew		3	ns
t_{PD}	Standard Bus Propagation Delay		3	
t_{AS}	Address Set-Up Time (at Slave Board)	50		ns
t_{DS}	Write Data Set Up Time	50		ns
t_{AH}	Address Hold Time	50		ns
t_{DHW}	Write Data Hold Time	50		ns
t_{DXL}	Read Data Set Up Time To XACK	0		ns
t_{DHR}	Read Data Hold Time	0	65	ns
t_{XAH}	Acknowledge Hold Time	0	65	ns
t_{XACK}	Acknowledge Time	0	8	μ s
t_{CMD}	Command Pulse Width	100	9.5	ns
t_{ID}	Inhibit Delay	0	100 (Recommend < 100 ns)	ns
t_{XACKA}	Acknowledge Time of of an Inhibited Slave	$t_{IAD} + 50$ ns	1500	
t_{XACKB}	Acknowledge Time of an Inhibiting Slave	1.5	8	μ s
t_{IAD}	Acknowledge Disable from Inhibit (An internal parameter on an inhibited slave; used to determine t_{XACKA} Min.)	0	100 (arbitrary)	ns
t_{AIZ}	Address to Inhibits High Delay		100	ns
t_{INTA}	INTA / Width	250		ns
t_{CSEP}	Command Separation	100		ns

APPENDIX B (Continued)
BUS TIMING SPECIFICATIONS SUMMARY

Parameter	Description	Minimum	Maximum	Units
tBREQL	↓BCLK / to BREQ / Low Delay	0	35	ns
tBREQH	↓BCLK / to BREQ / High Delay	0	35	ns
tBPRNS	BPRN / to ↓BCLK / Setup Time	22		ns
tBUSY	BUSY / delay from ↓BCLK /	0	70	ns
tBUSYS	BUSY / to ↓BCLK / Setup Time	25		ns
tBPRO	↓BCLK / to BPRO / (CLK to Priority Out)	0	40	ns
tBPRNO	BPRN / to BPRO / (Priority In to Out)	0	30	ns
tCBRO	↓BCLK / to CBRQ / (CLK to Common Bus Request)	0	60	ns
tCBRQS	CBRQ / to ↓BCLK / Setup Time	35		ns
tXCD	XACKI to Command Delay	0	1500	ns
tBSYO	CBRQ/I and BUSY/I to BUSY/I		12	μs
tCCY	C-clock Period	100	110	ns
tCW	C-clock Width	0.35 tCCY	0.65 tCCY	ns
tINIT	INIT / Width	5		ms
tINITS	INIT / to MPRO / Setup Time	100		ns
tPBD	Power Backup Logic Delay	0	200	ns
tPFINW	PFIN / Width	2.5		ms
tMPRO	MPRO / Delay	2.0	2.5	ms
tACLOW	ACLO / Width	3.0		ms
tPFSRW	PFSR / Width	100		ns
tTOUT	Timeout Delay	5	∞ (D.C.)	ms
tDCH	D.C. Power Supply Hold from ALCO /	3.0		ms
tDCS	D.C. Power Supply Setup to ACLO /	5		ms

APPENDIX C
BUS DRIVERS, RECEIVERS, AND TERMINATIONS

Driver 1,3						Receiver 2,3				Termination		
Bus Signals	Location	Type	IOL Min _{ma}	IOH Min _{μa}	CO Max _{pf}	Location	IIL Max _{ma}	IIH Max _{μa}	CI Max _{pf}	Location	Type	R Units
DAT0/→DATF/ (16 lines)	Masters and Slaves	TRI	16	-2000	300	Masters and Slaves	-0.8	125	18	1 place	Pullup	2.2 KΩ
ADR0/-ADRB/, BHEN/ (21 lines)	Masters	TRI	16	-2000	300	Slaves	-0.8	125	18	1 place	Pullup	2.2 KΩ
MRDC/,MWTC/	Masters	TRI	32	-2000	300	Slaves (Memory; memory- mapped I/O)	-2	125	18	1 place	Pullup	1 KΩ
IORC/,IOWC/	Masters	TRI	32	-2000	300	Slaves (I/O)	-2	125	18	1 place	Pullup	1 KΩ
XACK/	Slaves	TRI	32	-2000	300	Masters	-2	125	18	1 place	Pullup	510 Ω
INH1/,INH2/	Inhibiting Slaves	OC	16	—	300	Inhibited Slaves (RAM, PROM, ROM, Memory- Mapped I/O)	-2	50	18	1 place	Pullup	1 KΩ
BCLK/	1 place (Master us)	TTL	48	-3000	300	Master	-2	125	18	Mother- board	To +5V To GND	220 Ω 330 Ω
BREQ/	Each Master	TTL	5	-200	60	Central Priority Module	2	50	18	Central Priority Module (not req)	Pullup	1 KΩ
BPRO/	Each Master	TTL	5	-200	60	Next Master in Serial Priority Chain at its BPRN/	-1.6	50	18	(not req)		
BPRN/	Parallel: Central Priority Module Serial:Prev Masters BPRO/	TTL	5	-200	300	Master	-4	100		(not req)		
BUSY/, CBRQ	All Masters	O.C.	20	—	300	All Masters	-2	50	18	1 place	Pullup	1 KΩ
INIT/	Master	O.C.	32	—	300	All	-2	50	18	1 place	Pullup	2.2 KΩ
CCLK/	1 place	TTL	48	-3000	300	Any	-2	125	18	Mother- board	To +5V To GND	220 Ω 330 Ω
INTA/	Masters	TRI	32	-2000	300	Slaves (Interrupting I/O)	-2	125	18	1 place	Pullup	1 KΩ
INT0/→INT7/ (8 lines)	Slaves	O.C.	16	—	300	Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFSR/	User's Fron Panel?	TTL	16	-400	300	Slaves, Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFSN/	Power Back- Up Unit	TTL	16	-400	300	Masters	-1.6	40	18	1 place	Pullup	1 KΩ
ACLO	Power Supply	O.C.	16	-400	300	Slaves, Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFIN/	Power Back- Up Unit	O.C.	16	-400	300	Masters	-1.6	40	18	1 place	Pullup	1 KΩ
MPRO/	Power Back- Up Unit	TTL	16	-400	300	Slaves Masters	-1.6	40	18	1 place	Pullup	1 KΩ

APPENDIX C (Continued)
BUS DRIVERS, RECEIVERS, AND TERMINATIONS

Driver 1,3						Receiver 2,3			Termination				
Bus Signals	Location	Type	I _{OL} Min _{ma}	I _{OH} Min _{μa}	C _O Max _{pf}	Location	I _{IL} Max _{ma}	I _{IH} Max _{μa}	C _I Max _{pf}	Location	Type	R	Units
Aux Reset/	User's Front Panel?	Switch to GND (Note 5)	—	—	—	Masters	-2	50	18	None			
Notes:													
1. Driver Requirements													
I _{OH} = High Output Current Drive													
I _{OL} = Low Output Current Drive													
C _O = Capacitance Drive Capability													
TRI = 3-State Drive													
O.C. = Open Collector Driver													
TTL = Totem-pole Driver													
2. Receiver Requirements													
I _{IH} = High Input Current Load													
I _{IL} = Low Input Current Load													
C _I = Capacitive Load													
3. TTL low state must be $\geq -0.5v$ but $\leq 0.8v$ at the receivers													
TTL high state must be $\geq 2.0v$ but $\leq 5.5v$ at the receivers													
4. For the iSBC 80/10 and the iSBC 80/10A use only a 1K pull-up resistor to +5v for BCLK/ and CCLK/ termination.													
5. Recommend a 47Ω resistor in series with switch.													

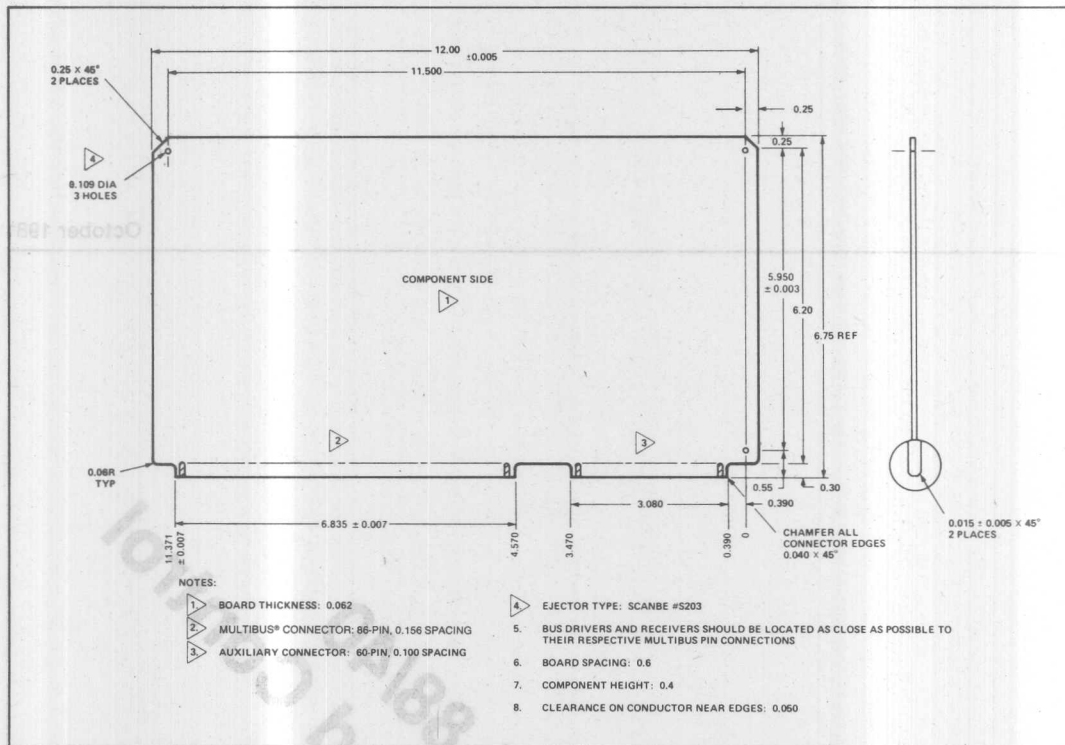
APPENDIX D

BUS POWER SPECIFICATIONS

	Standard (P1)				Optional (P2)					
					Analog Power		Battery Power Backup			
	Ground	+ 5	+ 12	- 12	+ 15	- 15	+ 5	+ 12	- 12	- 5
Mnemonic	GND	+ 5V	+ 12V	- 12V	+ 15V	- 15V	+ 5B	+ 12B	- 12B	- 5B
Bus Pins	P1 + 1,2, 11,12, 75,76 85,86	P1 + 3,4, 5,6,81, 82,83, 84	P1 + 7,8	P1 + 79, 80	P2 + 23, 24	P2 + 25, 26	P2 + 3,4, 5,6	P2 + 11, 12	P2 + 15, 16	P2 - 7,8
Nominal Output	Ref.	+ 5.0V	+ 12.0V	- 12.0V	+ 15.0V	- 15.0V	+ 5.0V	+ 12.0V	- 12.0V	- 5.0V
Tolerance from Nominal ¹	Ref.	± 5%	± 5%	± 5%	± 3%	± 3%	± 5%	± 5%	± 5%	± 5%
Ripple (Pk-Pk) ²	Ref.	50 mV	50 mV	50 mV	10 mV	10 mV	50 mV	50 mV	50 mV	50 mV
Transient Response Time ³		500 μ s	500 μ s	500 μ s	100 μ s	100 μ s	500 μ s	500 μ s	500 μ s	500 μ s
Transient Deviation ⁴		± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%
NOTES: 1. Tolerance is worst case, including initial voltage setting line and load effects of power source, temperature drift, and any additional steady state influences. 2. As measured over any bandwidth not to exceed 0 to 500 kHz. 3. As measured from the start of a load change to the time an output recovers within ± 0.1% of final voltage. 4. Measured as the peak deviation from the initial voltage.										

AP-28A

APPENDIX E MECHANICAL SPECIFICATIONS





APPLICATION NOTE

AP-114

October 1981

Using the iSBC™ 88/40 Measurement and Control Applications

Peter Andersen
OMS Applications Engineering

Order Number: 210263-001

INTRODUCTION

During the past twenty years, the automated process control industry has matured significantly. This is due to the introduction of the digital computer as an element of the control system. At the beginning of this period, the use of the digital computer was limited to a supervisory status in which the actual control was performed by various combinations of relay, analog, and pneumatic systems. Today, systems are off-the-shelf digital hardware and software to perform all the control applications. Indeed, the use of the hardware/software combination has opened entirely new areas of control applications.

The significant increase in computer capabilities and the corresponding reduction in size has been accompanied by a substantial drop in cost. This has led to a strong incentive for users to employ computers in totally new application areas which have resulted from this change in economics. Twenty years ago, few computer control projects were initiated and those which were could only be justified economically in terms of control systems which controlled upwards of 100 loops. Today, a microcomputer system can be justified for a small process which contains as few as 3 or 4 control loops.

Today, the control system engineer's decision is not so much an economic justification of a digital process as it is a choice of whether to use a single or a multiple microcomputer based design.

The trend toward the use of digital technology in the control world has been driven, in part, by the products which have been introduced into the marketplace by Intel Corporation. A recently announced product, the iSBC 88/40 Measurement and Control Computer, is in-

tended to further simplify the implementation of digital technologies into varied control applications and is the subject of this application note. Its architecture is well suited for both single microcomputer and multicomputing environments. The board is also easily adapted to a wide variety of input/output configurations through on-board facilities and iSBX MULTIMODULE expansion boards.

Generalized Computer Application Areas

Those applications in which computers are finding acceptance can generally be broken down into two broad areas. The first involves the acquisition and manipulation of process data by the computer, and is sometimes referred to as being a class of passive applications. The second, known as active systems, also involves the manipulation of the process itself. The systems in the latter class also provide various degrees of passive data manipulation.

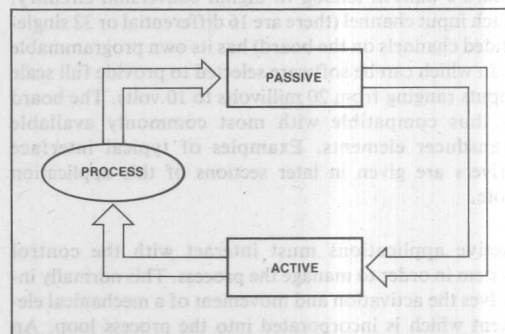


Figure 2. Classes of Computer Applications

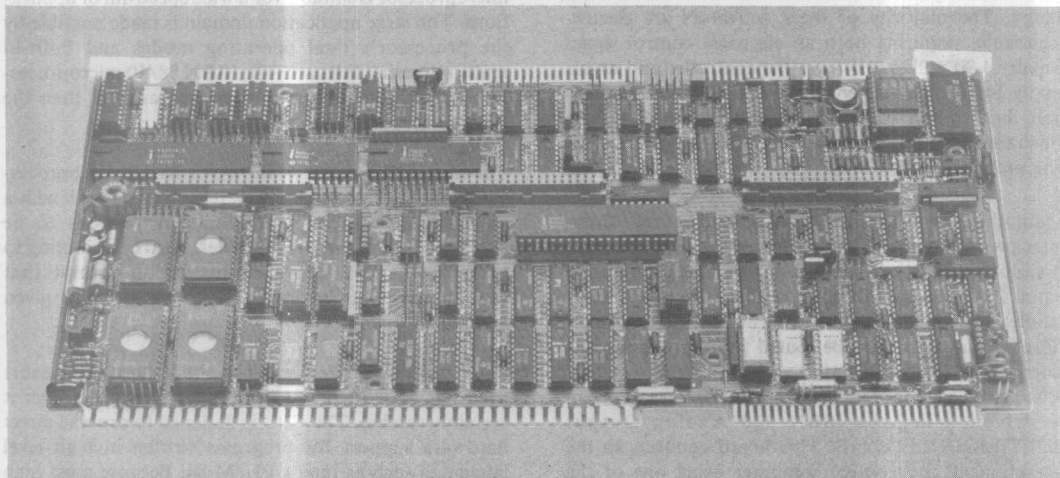


Figure 1. The iSBC™ 88/40 Measurement and Control Computer

At first glance, the area of passive computer applications seems to have little or nothing to do with process control; however, many computer design projects are being split into two phases. One phase is to characterize the process and the second is concerned with the actual control system. Many designs never move from phase one and are used as data acquisition systems.

The majority of passive systems involve measuring physical parameters of the process application. Examples are the measurement of pressure, temperature, flow, force, and level. Most transducers associated with these physical parameters provide an analog signal which is proportional to the physical property being sensed. Thus, the ability to measure analog voltages is a requirement of process control systems, both active and passive.

The iSBC 88/40 Measurement and Control Computer is ideally suited for these classes of systems because of the board's built-in analog to digital conversion circuitry. Each input channel (there are 16 differential or 32 single-ended channels on the board) has its own programmable gain which can be software selected to provide full scale inputs ranging from 20 millivolts to 10 volts. The board is thus compatible with most commonly available transducer elements. Examples of typical interface drivers are given in later sections of this application note.

Active applications must interact with the control system in order to manage the process. This normally involves the activation and movement of a mechanical element which is incorporated into the process loop. An amplifier and transducer are required to convert the electrical output of the controller into mechanical energy. The majority of these activators are electro-pneumatic, requiring both an electrical control signal (usually 4-20 milliamps) from the controller and an air supply for its internal pneumatic amplifier. Less common, but still in substantial numbers, are activators which use either a frequency input control signal or stepping motors.

Again, the iSBC 88/40 Measurement and Control Computer provides features designed to allow easy interface to various control actuators. For those actuators using digital frequencies or stepping motors, the board has a parallel output capability to drive up to 24 digital lines. Pulse output signals can be routed from programmable timers/counters (to generate a variety of pulse type outputs) to the external I/O devices. Analog actuators can be driven using the iSBX 328 Analog Output MULTIMODULE Board. This board connects to the measurement and control computer using one of the three iSBX connectors on the iSBC 88/40 board. Each

iSBX 328 board can generate up to 8 analog output signals, each of which can function in either a voltage or current (4-20 milliamps) output mode.

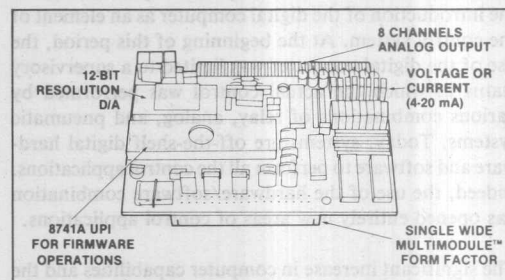


Figure 3. iSBX™ 328 Analog Output MULTIMODULE™ Board

Computer Processing Capabilities

The key to the rapid growth of digital computers in process control has been the flexibility offered by the software. The same hardware can be used in widely varying applications by allowing customization through software programming. To be successful in the process control marketplace, a digital computer system must be designed in a manner which optimizes the hardware/software relationships. The iSBC 88/40 Measurement and Control Computer does this well.

A powerful instruction set is mandatory if operations are to be efficiently performed by the processor. An instruction set optimized to perform business operations will perform poorly in an industrial process application. The processor used on the iSBC 88/40 board is the Intel iAPX 88/10 microprocessor. This third generation microprocessor is suitable for a wide spectrum of applications. The large application domain is made possible by the processor's dual operating modes and built-in multiprocessing features. The iAPX 88/10 microprocessor is from four to six times more powerful than the 8080A microprocessor.

The high performance of the iAPX 88/10 microprocessor is realized by combining an internal data path with a pipelined architecture that allows instructions to be prefetched during spare bus cycles. Also contributing to performance is a compact instruction format that enables more instructions to be fetched in a given amount of time.

Software for high-performance iAPX 88/10 processors need not be written in assembly language (although it certainly can be). The CPU is designed to provide direct hardware support for programs written in high level languages such as Intel's PL/M-86. Because most high level languages store variables in memory, the instruc-

tion set supports direct operation on memory operands, including operands on the stack. The hardware addressing modes provide efficient implementations of based variables, arrays, arrays of structures and other high level language data constructs. Hardware multiplication and division of signed and unsigned binary numbers, as well as unpacked decimal numbers, is fully supported by the CPU. In all, about 300 forms of machine level instructions are supported by the iAPX 88/10 processor.

Memory Options

A key design requirement for the iSBC 88/40 Measurement and Control Computer was to have the board support a variety of memory types and capacities. The result is a product which can easily be configured to meet a wide range of process control application requirements.

Program storage support for small to very large applications is obtained through the board's ability to include EPROM storage capacities ranging up to 64K bytes. Maximum standard storage capacity is from 8K bytes (using 2716 EPROM devices) to 32K bytes (using the 2764 EPROM). An optional EPROM expansion MULTIMODULE board can be mounted onto the iSBC 88/40 board to double the memory storage capacities.

Variables used in an application are usually stored in RAM memory. A standard on-board RAM capacity of 4K bytes is included on the measurement and control computer. In order to efficiently support multi-computer system design, 1K bytes of this memory is dual-ported. Dual-porting introduces a three bus system architecture to system design. An on-board local bus creates a data path between the iAPX 88/10 CPU and its local RAM. Data paths to RAM located on other iSBC boards are provided by the facilities of the MULTIBUS system bus. Finally, a third bus provides a gateway into the local RAM by other MULTIBUS single board computers or bus masters. If additional RAM is required, a small MULTIMODULE RAM expansion board can be attached to the iSBC 88/40 board to add 4K bytes of random access memory.

Even more flexibility can be gained by using unneeded EPROM memory sockets. Because JEDEC standard 24/28 pin sockets have been used, byte wide RAM modules can be inserted into areas of the EPROM memory space. The use of this RAM can considerably enhance the design of certain applications. The board capabilities are such that it is not necessary to have all devices residing in the EPROM sockets be of the same type or size.

Many process control applications require the use of non-volatile memory for the storage of parameter lists

and system setpoints. Provision has been made on the iSBC 88/40 Measurement and Control Computer to fully support Intel's new 2816 Electrically Erasable and Programmable Read Only Memory (E²PROM). This device gives the user 2K bytes of memory. Depending on the application, up to eight devices (16K bytes) can be used on the iSBC 88/40 board. The board includes all required voltages and wave-shaping circuits to fully support the use of the 2816. A byte of 2816 memory can be programmed in 16 milliseconds. A subsequent section of this application note contains a comprehensive discussion of the operation of the board with the 2816.

Arithmetic Functions

Using computers as an element in a control system leads to extensive arithmetic and mathematical functions. To be effective and attractive to the designer, a computer board must provide a wide range of mathematical capabilities. The iSBC 88/40 Measurement and Control Computer easily meets these needs with varying capabilities for hardware and software functions.

Many applications are adequately handled using the hardware add/subtract and multiply/divide instructions of the on-board iAPX 88/10 processor. Functions needing integer arithmetic of varying precisions are easily programmed using this facility. In some cases, more complex operations may require the use of software libraries to gain the required mathematical functions. The speed and instruction set of the CPU, in conjunction with PL/M-86 statements, make programmers comfortable with these operations.

As processes become more involved and their control algorithms more complex, the need for the processor to support more precise numbers becomes important. The additional precision is usually obtained through the use of a floating point representation. Intel supplies several tools which simplify the implementation of systems requiring floating point operations. Complete support for the floating point numbers is provided as an integral part of the PL/M-86 compiler. Thus, variables can be specified as real numbers. The compiler will perform all numerical operations on these numbers in the floating point format. The data formats of all Intel floating point support conform to the proposed IEEE Floating Point Standard, insuring highly accurate results.

An important feature of the iAPX 88/10 processor is its ability to use a co-processor. The Intel 8087 is mounted on the iSBC 337 MULTIMODULE Numeric Data Processor to provide arithmetic and logical instruction extensions to the 8086 and 8088 CPU's. The instruction set consists of arithmetic, transcendental, logical, trigonometric, and exponential instructions which can all

operate on seven different data types. In many cases, the use of this MULTIMODULE board results in two orders of magnitude performance enhancement over a software solution. This board is the subject of a subsequent section of this application note.

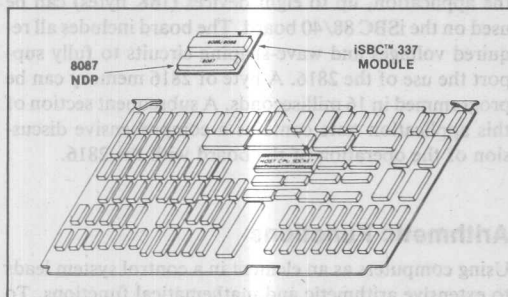


Figure 4. ISBC™ 337 MULTIMODULE™ Numeric Data Processor

APPLICATION EXAMPLE

The features of the ISBC 88/40 Measurement and Control Computer can best be shown through an example. This application note describes the classical control system application of an agitated heating tank. Figure 5 shows the prominent features of this process control applications. The process consists of a storage vessel, a temperature sensor which measures the temperature of the fluid leaving the vessel, and a steam coil whose steam flow is regulated by a proportional valve. A motor drives an agitator to insure the temperature of the tank remains homogeneous.

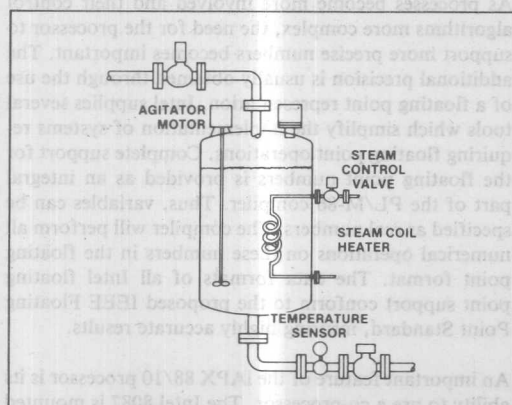


Figure 5. An Agitated Heating Tank

The passive portion of the application involves measuring the actual temperature of the fluid as it leaves the tank (and thus the temperature of all the fluid in the

tank). If a control system is to be constructed which will control the temperature, an algorithm must be implemented which will provide control of the steam valve based upon the actual and the desired temperatures. This is the active portion of the application.

The control algorithm selected to control the tank temperature must be capable of compensating for disturbances created by a variety of conditions. For example, the temperature can be affected by changes in steam temperature, input temperature of the fluid, output flow rate, ambient temperature, and the flow rate of steam through the steam coil. Our control system will have control of only one of the variables and will only monitor the output temperature. To gain a degree of stability under these conditions, a feedback control algorithm is required. Alternatively, a system could be implemented using a feed-forward control algorithm. Unfortunately, the latter technique would require extensive instrumentation of all possible variables which could cause a disturbance. A feedback control system can take corrective action regardless of the source of a disturbance. Its chief drawback is that no corrective action is taken until an error is actually detected and, if not "tuned" correctly, some oscillations can occur.

Classical Controller Approaches

Before proceeding with a discussion of how a control system can be implemented using single board computers, a short discussion of classical control system theory is in order. This material will provide a background into the control algorithms which will be used as a basis for the digital control solution which will be developed.

The classical controller for feedback systems uses the "three mode" or PID (Proportional, Integral, Derivative) algorithm. In this system, the control output signal is a function of the error (the difference between the set-point and the measured system variable). A specific application will use some combination of one, two, or all three terms making up the control statement.

Before continuing with the implementation of the control algorithm on the ISBC 88/40 Measurement and Control Computer, the various terms of the equation will be reviewed.

For Proportional control, the controller output is given by the equation:

$$m(t) = b + k_0 e(t) \quad (\text{eq. 1})$$

where $m(t)$ is the output signal, b is an adjustable bias value, k_0 is a gain constant, and $e(t)$ is the measured error signal. Proportional control systems are normally

not used by themselves since corrections can not be made until an appreciable error has been detected. In addition, they tend to introduce oscillations into the system if the gain is set too large. Another disadvantage of proportional only systems is their inability to maintain a control element at some point (other than at its zero point using the bias term) in the absence of an error signal.

The second term in the PID solution is the Integral. The result of this term is to eliminate steady-state error or offset. The elimination of the offset is an important control objective; thus, the integral control term is widely used in conjunction with the proportional control element. The equation for the integral term is:

$$m(t) = (1/k_I) \int e(t) dt \quad (\text{eq. 2})$$

where k_I is the integral or reset time.

The Derivative term in the algorithm is used to provide an output which is a function of the rate of change in the error signal. It anticipates the future behavior of the system and improves the dynamic response to the controlled variable by decreasing the process response time. The format for the derivative term is:

$$m(t) = k_D (de/dt) \quad (\text{eq. 3})$$

where k_D is a constant representing the derivative time expressed in seconds or minutes. Because the output of the term is zero for a constant error, derivative control is never used alone in a control system. Instead, it is always used in conjunction with proportional and integral control. The derivative term is seldom used in flow controllers because derivative control tends to amplify "noise" which is picked up in the flow measurement, leading to an unstable control system. In addition, systems which have very large time delays do not benefit from the use of this term.

Implementation Using Digital Techniques

With an exposure to the fundamental concepts of control theory complete, the development of a solution using the iSBC 88/40 Measurement and Control Computer can proceed. A modular "top-down" approach will be used in this application note. The general requirements will be defined and "black boxes" will be developed to meet these requirements. Finally, the individual pieces will be combined to form a complete solution to the agitated tank control problem.

An effective control algorithm must deal not only with the mathematical solution of the control equation, but must also provide tests on limits and error conditions.

As this application note will show, the iSBC 88/40 Measurement and Control Computer is easily able to support these additional requirements.

Additional supporting functions are also needed to effectively implement a complete control system solution. For example, provisions must be made to support input and update of the controller setpoints. Allowances must be made to modify control algorithm constants in order to "fine tune" the system after start-up. Raw analog data must be filtered to eliminate spurious sensor measurements and then must be converted into engineering units. In earlier system implementations not based on digital computers, these functions were performed using a "black box" approach. Here, each function is considered separately and the final solution is composed of combinations of building blocks.

Digital technology offers a simple analogy to this approach. Because application design is performed with software, a "black box" design is available for use with microcomputers. The black box corresponds to a software "task" and the system is integrated into a functional unit using a real time operating system. The iRMX 88 Real Time Executive provides all the tools needed by the software designer to implement his required functions for the application. This application note will show how the iRMX 88 executive can be used to simplify the design and to provide significant features in a process design example.

Figure 6 shows a block diagram of the operations needed to implement the control of one loop for the agitated heating tank. An attribute of using digital microcomputers is that additional loops can be run using the same hardware and software until the I/O or processing capabilities have been exceeded.

Each element of the block diagram represents one function which must be performed by the system. A task will be written to perform the functions assigned to each block. When the tasks are configured together with the iRMX 88 executive, a complete control solution will result. Some key features of the iSBC 88/40 Measurement and Control Computer will now be examined and a typical implementation will be described.

ANALOG SUPPORT FUNCTIONS

The information presented in Figure 6 indicates that many functions involve the manipulation of analog data and its conversion into a digital form usable by the processor. This involves the use of both hardware and software. This section of the application note demonstrates how the iSBC 88/40 board features can be applied to the solution of the analog portions of the system implementation. Both software programming concepts and hardware support products are examined.

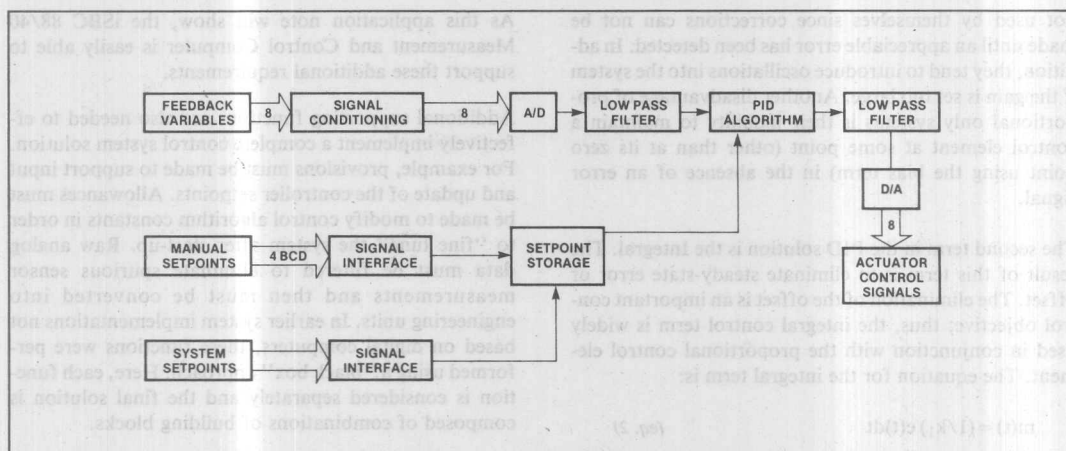


Figure 6. Control System Block Diagram

A digital computer performs most of the control system operations using software. Data is sampled from the process sensor and converted to an equivalent digital format. Subsequent operations use the digital form of the data. Unfortunately, this requirement for operating on sampled data, rather than continuous actual data, can lead to errors if the system is not properly implemented. Care must be taken to minimize errors when the original signal is digitized. Figure 7 shows how the digital signal may look when an analog signal is sampled using an analog to digital converter. A glance at the figure indicates that the error can be minimized by taking samples at shorter time intervals so that the staircase more closely resembles the original signal. Indeed, this is true, but what sample rate is best for a particular input signal?

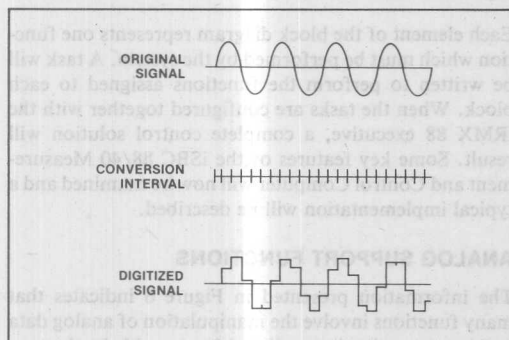


Figure 7. Analog Signal Digitization

A rule for digital control system designers is that the sample must be performed more than twice each period of the original analog signal. Thus, the sampling period must be less than one half the period of the sinusoidal

frequency component which must be digitized. Even this method does not, in itself, assure an accurate measurement. Figure 8 shows the effects of the aliasing phenomenon on a high frequency signal. Aliasing converts the high frequency components into fictitious low frequency signals in the sampled results. Before data obtained from a digital system can be used, the unwanted signals must be filtered from the original sensor signal.

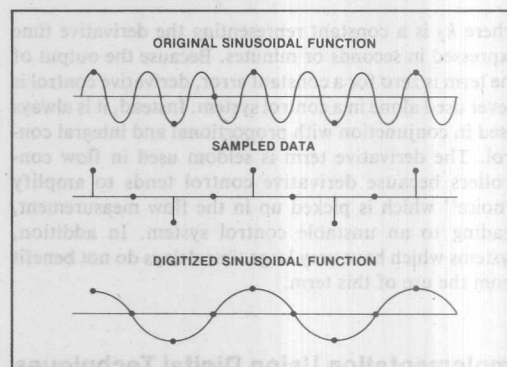


Figure 8. High Frequency Aliasing Error

Two approaches can be considered for filtering the data. One is the creation of an analog low pass filter and the second is the implementation of a digital filter. Unfortunately, a digital filter cannot remove aliasing error and is normally used to provide filtering of very low frequency oscillations. Analog filtering provides effective removal of unwanted frequencies but is expensive when attempting to gain sharp cut-off frequencies. A combination of the two technologies results in an ideal situation when used with digital controllers such as the iSBC 88/40 Measurement and Control Computer.

The final choice of sampling rate is usually determined by examining the process to be controlled. If a mathematical first order transfer function can be obtained for the process, either theoretically or experimentally, then the choice should be to use one tenth of the process time constant. If no function can be obtained and the frequency of the input signal is known and bounded, a sample rate equal to at least twice the input frequency is used. If none of the above is known, a rough estimate for process applications is to use a 1 second sample period for flow measurements, a 5 second interval for level or pressure measurements, and a 20 second interval for temperature or composition measurements. In any case, faster sampling than is necessary is a waste of computing power and limits the number of PID loops that can be supported by a given system.

The elimination of high frequency noise in systems using Intel's control products is best accomplished using the iCS 910 Analog Termination Strip. This strip has provision for the installation of a single pole RC low pass filter (details on the use of this strip in industrial control applications can be found in AP-52, Using Intel's Con-

trol Series In Industrial Applications). In addition to providing a front-end low pass filter, the strip gives a simple method of terminating analog wiring to the analog to digital converter. Figure 10 indicates the cable connections which can be used to connect the analog input connectors of the iSBC 88/40 board to the iCS 910 termination strip. This connection arrangement will provide complete compatibility between the numbered channels on the termination strip and those defined by the measurement and control computer.

The iSBC 88/40 board's application software can be used to eliminate the effects of low frequency noise in the sampled signal. This is done by implementing a simple digital low pass filter. The equation for a first order filter is:

$$Sf = a(Sm) + (1 - a)(Sf') \quad (eq. 4)$$

where Sf represents the filtered output, a is a function of the cutoff frequency, Sm is the measured sample, and Sf' is the last filtered output result. If additional poles are required, the equations can be cascaded as required.

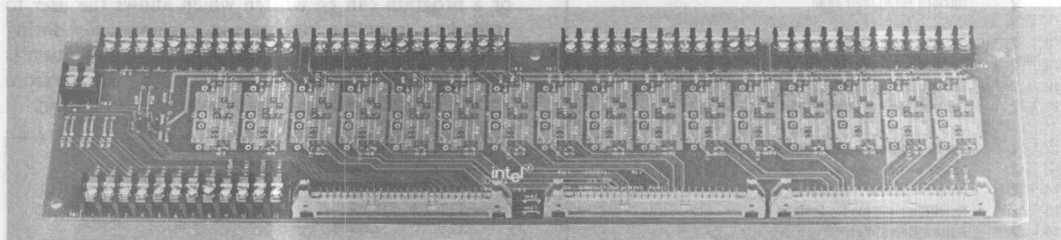


Figure 9. iCS™ 910 Analog Termination Strip

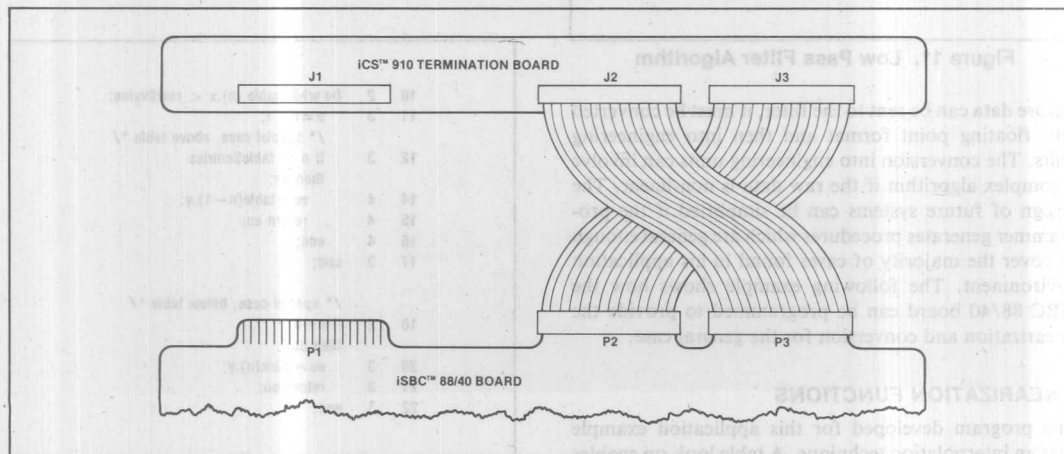


Figure 10. Termination Board Interconnects

The implementation of this filter using Intel's PL/M-86 high level language is straightforward. A simple procedure can be written in which the measured value, the last filter output value, and the value for α are passed with the call. The procedure returns the new filtered value. The code for such a procedure is shown in Figure 11. Note that the computation is performed in steps to prevent any stack overflows from occurring when real numbers are used. This should be done whenever the algebraic equation exceeds eight terms. The 8087 stack used in internal operations can overflow when more than eight operations are nested together. Breaking the equation into smaller steps can prevent any overflow errors from occurring.

```

1      Analog$filter$module: Do;
2  1      Analog$filter:
3      2      Procedure (present$value, last$output, cutoff) real public;
4      2      Declare (present$value, last$output, cutoff) pointer;
5      2      Declare New$signal based present$value real;
6      2      Declare Old$filter based last$output real;
7      2      Declare Alpha based cutoff real;
8      2      Declare New$filter real;
9      2      Declare temp1 real;
10     2      Declare temp2 real;
11     2      Declare temp3 real;
12     2      Declare One real data (1.0);
13     2      temp1 = Alpha * New$signal;
14     2      temp2 = One - Alpha;
15     2      temp3 = temp2 * Old$filter;
16     2      New$filter = temp1 + temp3;
17     2      Return New$filter;
18     2      end Analog$filter;
19     2      end Analog$filter$module;

```

Figure 11. Low Pass Filter Algorithm

Before data can be sent to the filter, it must be converted into floating point format and then into engineering units. The conversion into engineering units can involve a complex algorithm if the raw data is non-linear. The design of future systems can be simplified if the programmer generates procedures which are general enough to cover the majority of cases found in his application environment. The following example shows how the iSBC 88/40 board can be programmed to provide the linearization and conversion for the general case.

LINEARIZATION FUNCTIONS

The program developed for this application example uses an interpolation technique. A table look-up enables a program to be written which will support both linear

and non-linear analog sensors. The number of entries in the table is a function of the desired resolution and of the non-linearity. For example, linear functions needing only scaling and offset ($y = ax + b$) require only two table entries. A separate table is maintained for each sensor channel. The program is written to support a maximum of 256 entries per channel which should provide at least 0.1 percent accuracy for all but the most non-linear applications.

Each table entry consists of a raw value and a corresponding real engineering unit value expressed in floating point format. The linearization program's declaration of such a table is shown in Figure 12. The application software must determine the bracket or location of the terms in the table which lie above and below the raw input value. The algorithm to find the bracket in the table which corresponds to the raw data input can be programmed as shown in Figure 13. Once the bracket has been found, the actual engineering value can be calculated and passed back to the calling program. The code for performing the interpolation calculation might look like that shown in Figure 14. Data for the tables can be determined from known characteristics of the sensor or a program can be written which allows the user to enter known points into the table dynamically during calibration. In this application note, an assumption is made that the data has been entered into the table from known characteristics rather than actual calibration.

```

6  2      Declare (table based table$pointer)(255) structure (
      x word,
      y real );

```

Figure 12. Declaration of Table

```

10 2      Do while table (n).x < raw$value;
11 3          n = n + 1;
12 3          /* special case, above table */
13 3          If n > table$entries
14 4              then do;
15 4                  eu = table(n-1).y;
16 4                  return eu;
17 3          end;

18 2          /* special case, below table */
19 2          If n = 0
20 3              then do;
21 3                  eu = table(n).y;
22 3                  return eu;

```

Figure 13. Bracketing Algorithm

```

/* interpolate engineering units */
23 2 dx = float(int(table(n).x - table(n-1).x));
24 2 dy = table(n).y - table(n-1).y;
25 2 dr = float(int(raw$value - table(n-1).x));
26 2 eu = dr * dy;
27 2 eu = eu / dx;
28 2 eu = eu + table(n-1).y;

```

Figure 14. Interpolation Algorithm

One final component of the analog design which is required is the creation of software which will actually interface with the analog to digital converter and transform data from the analog world into a digital domain. Again, a program should be developed which is general enough to handle a wide variety of applications. It should be compatible with both the on-board A/D sections and with the iSBC 311 Analog Input MULTIMODULE Board, which may be installed for analog expansion.

The interface with the analog portions of the boards is easily handled using software. The ADC can be commanded to select the desired analog channel and begin a conversion by sending the appropriate byte containing the channel and gain bits to a port corresponding to the ADC. When using the on-board converter, the iSBC 88/40 board user should send the command byte to port 0D8 hex. The actual selection of the desired channel and the conversion takes only 50 microseconds, so little is gained by using an interrupt instead of status testing to detect the end of conversion. The status bit is tested by reading the input status port (0D8 hex for the on-board converter). When the conversion is complete, the bit will have a value of 0.

Certain multiplexer components used in the ADC require that a delay time be added to the basic 50 microseconds for the channel to settle after a new gain setting has been selected before reading the sample and hold converter. The amount of delay is a function of the gain and varies from 0 (gain = 1) to 30 milliseconds (gain = 250). The analog driver software must take this settle time into account. Figure 15 shows the required settle times for the various gain settings. The delay is easily implemented using the facilities of the iRMX 88 nucleus. While the system is waiting for the settling time, other tasks can use the processor to execute their code. Figure 16 provides an example of a program which gets data from the analog to digital converter for a selected channel and gain. The iRMX 88 request for a time delay is implemented using the call to RQWAIT specifying the desired delay. In the example, the system delay increment is assumed to be 5 milliseconds, so the required number of delay increments is specified as 6 in order to wait for 30 milliseconds at high gains. Note that, for

gains of one, the delay is skipped. After the required delay has elapsed, the converter is again activated using another output to its command port. This output must again include the channel and gain information.

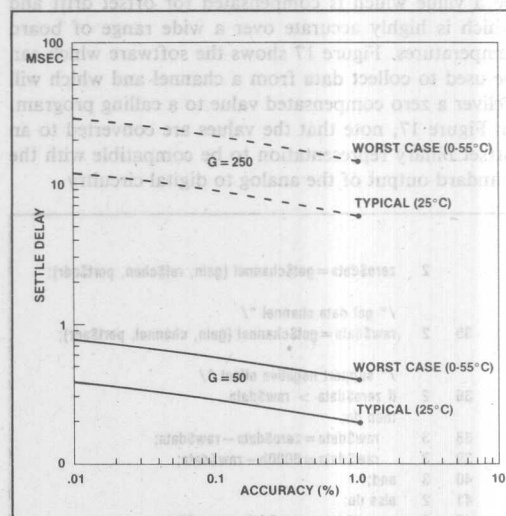


Figure 15. Analog Settle Times

```

/* select mux channel */
14 2 output(port$adr) = channel or gain;
15 2 if gain < 40h
    then do;
        /* settling delay for high gains */
17 3 msg$ptr = rqwait(.timeout, 6);
18 3 output(port$adr) = channel or gain;
19 3 end;
        /* wait for end of conversion */
20 2 do while (input(port$adr) and 01h) > 0; end;

        /* get adc data */
22 2 low$raw$data = input(port$adr) and 0f0h;
23 2 high$raw$data = input(port$adr + 1);
24 2 raw$data = shl(high$raw$data, 8) or low$raw$data;

```

Figure 16. Analog Input Routine

A workable analog driver must provide more than just the ability to get data from a specified channel. At a minimum, the zero offset induced by the temperature of the circuitry must be removed from the raw data. In some cases, an additional correction is required to compensate for gain error induced by temperature. However, the effect of the latter is small and can usually be ignored.

Provisions are included on the iSBC 88/40 Measurement and Control Computer to simplify the task of providing a zero offset correction. Wire-wrap stakes are mounted

on the board to facilitate grounding one of the input channels. In the differential mode of operation, channel 15 represents the zero reference offset voltage. If a data channel has the offset subtracted from it, the result will be a value which is compensated for offset drift and which is highly accurate over a wide range of board temperatures. Figure 17 shows the software which can be used to collect data from a channel and which will deliver a zero compensated value to a calling program. In Figure 17, note that the values are converted to an offset binary representation to be compatible with the standard output of the analog to digital circuitry.

```

2   zero$data = get$channel (gain, ref$chan, port$adr);

/* get data channel */
35  2   raw$data = get$channel (gain, channel, port$adr);

/* support negative offset */
36  2   if zero$data > raw$data
    then do;
38  3   raw$data = zero$data - raw$data;
39  3   raw$data = 8000h - raw$data;
40  3   end;
41  2   else do;
42  3   raw$data = raw$data - zero$data;
43  3   raw$data = raw$data + 8000h;
44  3   end;

```

Figure 17. Zero Compensation Procedure

The analog input driver required for the application can now be constructed using the software building blocks which have been created. Generally, the input data will consist of either thermocouple inputs or non-temperature sensitive inputs. The driver must be able to support both by providing a selective cold junction compensation correction for those channels which are designated as thermocouples.

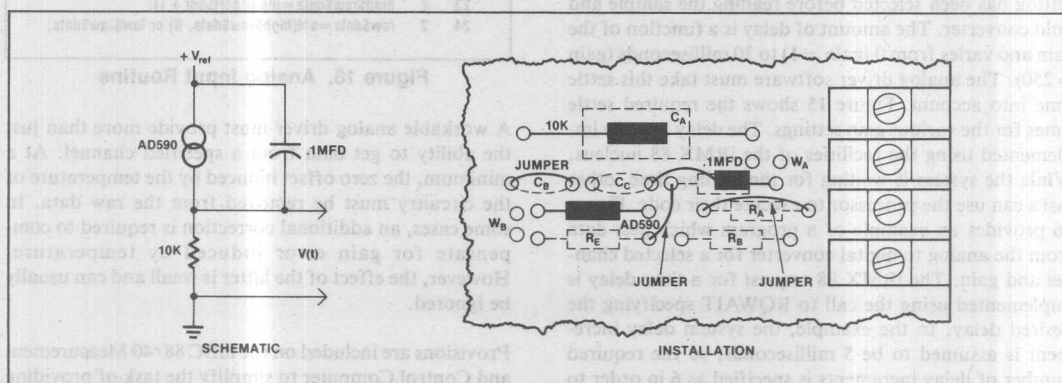


Figure 19. iCS™ 910 Board Sensing Circuit

The problem is illustrated in Figure 18. The voltage which represents the temperature of the thermocouple consists of the sum of the actual thermocouple voltage plus the voltage which is generated by the thermocouple junctions created where the wiring is terminated. The error introduced by the termination must be removed before a junction temperature can be calculated. If the thermo/voltage characteristics of the termination junction are known, the induced error can be subtracted and the temperature of the thermocouple can be calculated.

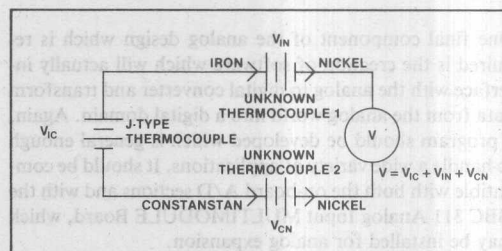


Figure 18. Thermocouple Cold Junction Error

Two things must be known for the correction voltage to become available. First, the actual temperature of the junction board must be known. Second, the electrical characteristics of the junction with respect to temperature must be defined. With this data available, the correction voltage can be obtained using the linearization program which has been created as an analog building block.

The first problem is solved by installing a temperature sensing circuit onto the iCS 910 Analog Termination Strip. Figure 19 shows such a circuit which can be used to provide an extremely accurate measurement of the board and terminator temperature. Note that the circuit is installed onto the termination board using the mounting locations originally designed for the installation of a low pass filter. The output of the temperature sensing is

related only to the temperature of the sensor device which provides a current of 1 microamp per degree Kelvin through the 10K resistor. The temperature is related to the voltage by the equation:

$$V = (273 + T) / 100 \quad (\text{eq. 5})$$

Thus, the voltage read from the termination strip as the temperature varies from 0 to 70 degrees Centigrade will vary from 2.73 volts at 0 degrees to 3.43 volts at 70 degrees. The analog to digital converter should operate at a gain of one to read this voltage. This will provide a resolution (1 bit change) of 0.70 volts / 0.00244141 volts/bit or 286 bits/70 degree change. This equates to about 0.25 degree per bit change.

The second problem is solved by connecting a thermocouple, which is placed in an ice bath, to the iCS 910 strip. The strip is placed into an environmental chamber and the output monitored as the board and junction temperature is varied. The output represents the correction required at each temperature. Tests made for this application note indicated that the error was essentially linear over the board range from 0 to 70 degrees Centigrade. The correction voltage was found to vary linearly from minus 0.102 millivolt at 0 degrees to 3.578 millivolts at 70 degrees. This data was placed into a linearization table to give an offset correction for a measured temperature of the board. Figure 20 shows the table and the code required to correct the raw temperature value from thermocouple inputs.

```

60 7 /* get thermocouple reference junction temp */
    j$raw = analog$to$digital$conversion (
        gain$one,
        13,
        channel$data, port$number );

61 7 tc = analog$linearization (
    @cold$junction$stable,
    2,
    j$raw );

62 7 tc = analog$filter (
    @tc,
    @channel$data, last$thermocouple,
    @channel$data, filter$cuttoff );

63 7 raw = raw + unsign(fix(tc));
    7 channel$data, last$thermocouple = tc;

```

Figure 20. Thermocouple Correction Program

An analog input driver can now be constructed which is compatible with a variety of applications. It will run as a task under the iRMX 88 nucleus. In order to support up to "n" analog inputs, an exchange is used to store information about current active analog channels. User tasks requiring analog facilities send a request to the analog

exchange indicating the parameters of the desired channel. Because an exchange has a FIFO storage capacity for messages, each active channel is sampled by the task in turn, then placed back onto the exchange. A unique message is used to indicate the beginning of the channel requests. Figure 21 provides a partial listing of the code used to make up the analog input task.

```

57 3 msg$ptr = r$wait (.timeout, 2);

58 3 last$channel = false;

59 3 do while last$channel = false;

60 4 msg$ptr = r$wait (.analog$exch, 0);
61 4 if channel$data, type = null$type
    then last$channel = true;
63 4 else do;

    /* test for conversion time request */
64 5 if channel$data, conversion$counter = 0
    then do;

        /* get raw data from adc */
74 6 raw = analog$to$digital$conversion (
        gain,
        channel$number,
        port$number );

83 6 /* perform engineering unit conversion */
        eu = analog$linearization (
            table$pointer,
            number$of$entries,
            raw );

84 6 /* filter the data */
        eu = analog$filter (
            @eu,
            @channel$data, last$value,
            @filter$cuttoff );
85 6 channel$data, last$value = eu;

86 6 channel$data, conversion$counter
        = conversion$interval;

87 6 exch$ptr = channel$data, output$exchange$ptr;
88 6 data$ptr = r$wait (exch$ptr, 0);
89 6 data$message, value = eu;
90 6 call r$send (exch$ptr, data$ptr);

91 6 end;

    /* decrement counter if not ready yet */
92 5 else channel$data, conversion$counter =
        channel$data, conversion$counter - 1;

93 5 end;
94 4 call r$send (.analog$exch, msg$ptr);

```

Figure 21. Analog Input Task

Updated data is stored in an output exchange in order to assure mutual exclusion of the engineering unit conversion of the data. Mutual exclusion guarantees that the data cannot be read by another task while it is being up-

dated (during the updating process, multiple bytes of data must be changed; until all are modified, the number cannot be considered valid). The exchange mechanism of iRMX executives supports the movement of messages (this might be compared to a letter in a mailbox). If the data is stored as a message in an exchange, it is available to the first user requesting it. While that user has the message (letter), it is not available to anyone else. When he is finished with it, he will return it to the exchange so that other users may operate upon the data. Note that the sample interval of each channel is selected by the requesting task so that optimum processor efficiency can be obtained.

Certain parameters used by the analog input task must be retained even if the system power is shut off for an extended period of time. These parameters are used to provide the task with unique information such as the channel and port address, the desired gain, the conversion interval and the linearization and engineering conversion data. On the other hand, some information used

by the task can be easily created dynamically and does not require the use of non-volatile storage. Examples of the latter category include addresses of the storage exchanges and addresses of the various messages.

The use of E²PROM on the iSBC 88/40 Measurement and Control Computer provides the mechanism for the storage of those parameters which must be occasionally modified. Figure 22 shows a possible technique for passing the analog input task its required information and pointers to the non-volatile data. Intel's PL/M-86 provides a convenient mechanism for referencing variables whose physical location is passed as a parameter. This is the BASED VARIABLE. A declaration is made which indicates the location of the variable containing the address of the data. For example:

```
Declare CONSTANT$POINTER pointer;
Declare CONSTANT based
CONSTANT$POINTER real;
```

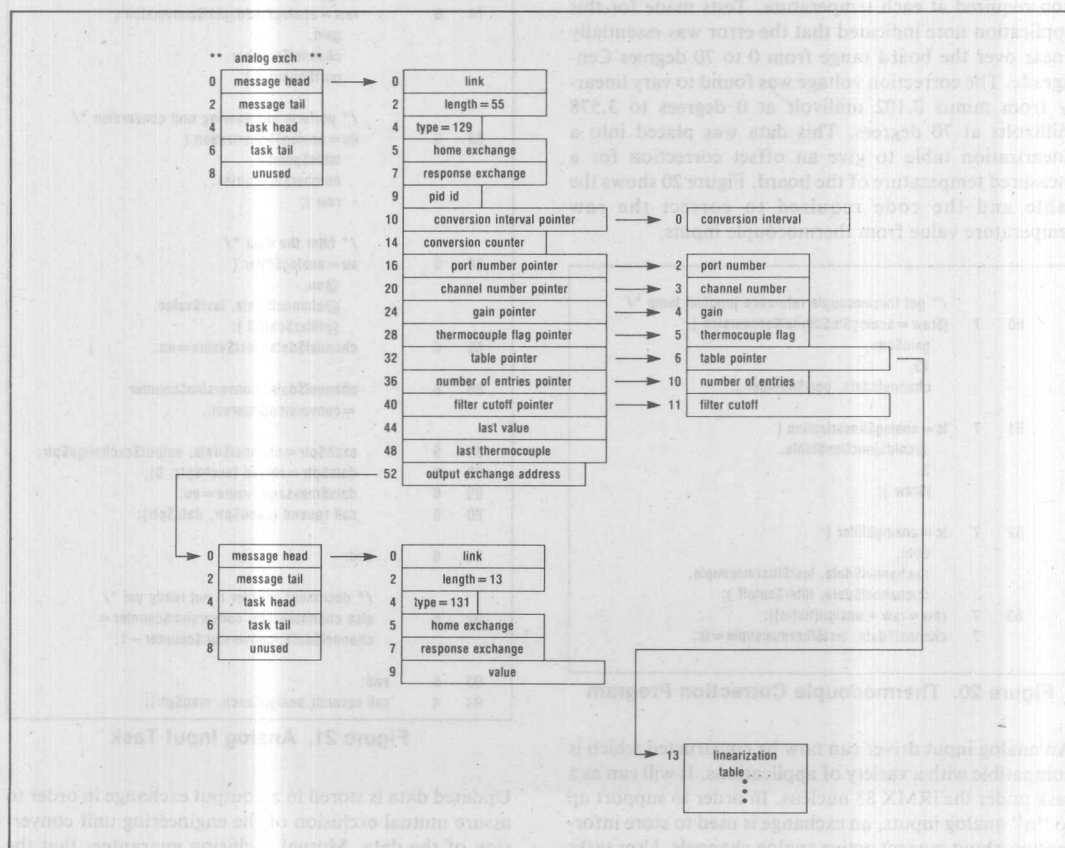


Figure 22. Analog Input Data Structures

The `CONSTANT$POINTER` contains the address of the constant which is to be used in the calculations. Any program reference to `CONSTANT` will cause the processor to use the real number stored in the address pointed to by `CONSTANT$POINTER`.

This technique allows a message to contain pointers to E²PROM constants which can be used by the task in performing its functions. Indeed, multiple levels of based variables can be used as shown in Figure 22.

CONTROL ALGORITHM

The implementation of a control algorithm on the iSBC 88/40 computer involves more than just implementing the PID equation. To become truly cost effective, multiple loops must be supported by the board and error checking/correction must be included. As with the analog input functions, system control parameters must be maintained in non-volatile memory. Finally, the system must be capable of operating in real time with a minimum of required processor time. This section examines some of the features which are used to provide these functions on the measurement and control computer.

The first design goal is to support multiple control loops using as little of the processor's time as possible. The processing time is minimized by performing all complex mathematical computations using the 8087 math co-processor mounted on the iSBC 337 MULTIMODULE board. Certain details of the software implications of the co-processor are important to the system designer.

In many cases, the iRMX 88 nucleus will provide all the required initialization operations for the co-processor chip. The nucleus sends a default control word setting the device to mask all exceptions and interrupts, define a 64 bit precision, and to round up all operations. In an iRMX environment, the application programmer has no need to send additional mode commands to the processor. However, the mode can be changed by using the PL/M built-in procedure, `SET$REAL$MODE`, if required. In the application code written for this application note, certain conversion algorithms required that results obtained from the math operations be truncated. To instruct the 8087 to perform this truncation, a command word of 0FBF hex is sent in the initialization segment of a task.

Multiple control loops are implemented using the iRMX 88 exchange mechanism. Here, messages are queued at an exchange in a first in, first out (FIFO) manner. One message can be sent to the exchange for each control loop to be executed. A special message is placed into the exchange at control task initialization to be used as a pointer to the end of the queue. Each time the control

task is to run, it will read messages sequentially from the exchange until it encounters its end of queue message. Each message corresponds to one control loop's specifications and is returned to the exchange when the loop has been completed. A separate control interface task manages the control loop activation by sending a message containing the necessary parameters to the control exchange. This technique allows the interface task to also remove a control task by taking the appropriate message from the exchange when parameter modifications or control loop deletion is requested.

Each message at the control exchange (in the application example, this exchange is called `PID$EXCH`) contains pointers to various other exchanges or data structures. The relationships of these structures is shown in Figure 23. Note that some system parameters should be stored in non-volatile E²PROM memory. System constants are stored in an exchange pointed to by the primary control message. An exchange is used here so that the system can provide mutual exclusion of the data if it is required to modify one or more of the parameters while the control system is running. Additional exchanges are used to store the input and output terms in order to insure compatibility with the analog input and output tasks.

In order to function correctly, a digital implementation of a PID control algorithm requires operation at a known time interval. In the case of the implementation constructed for this application note, a time increment of 100 milliseconds was desired. The iRMX nucleus provides the ability to perform a timed wait at an exchange via the call to the primitive procedure, `RQWAIT`. Unfortunately, this procedure can not be directly used in the task to provide the required task delay. This is because the execution time of the task is a function of the number of loops being implemented and also varies slightly depending upon the program paths required by the data values. Thus, a mechanism must be implemented to provide the task synchronization.

The desired time delay can easily be obtained by using an associated synchronization task. In this task, the `RQWAIT` primitive can be used with the required time delay. Because the task execution time is not a variable, this task can be used to provide synchronization for its supported task. Figure 24 shows how the two tasks can communicate with each other. Two exchanges are maintained. One, called the PID bucket exchange in the implementation, is used by the main task to indicate that it is beginning its execution and that a new time period delay is to begin. The timer task (whose priority should be greater, i.e., having a smaller priority number) will wait at the bucket exchange for the message. When it is received, it will begin a delayed wait at an exchange. When the timeout period has elapsed, the message is sent to a second exchange (in the figure, this exchange is

called the PID trigger exchange). The main task, after completing the servicing of all operational PID control loops, will wait at the trigger exchange for a message from the timer task. In the case of the application example, the message will arrive 100 milliseconds after the task began its last update of the control loops.

When iRMX 88 timed wait operations are implemented on the iSBC 88/40 Measurement and Control Computer, timer 0 of the on-board 8253 programmable interval timer must be used. A wire wrap jumper must be installed to vector the output of the timer to one of the interrupts of the 8259A programmable interrupt controller chip.

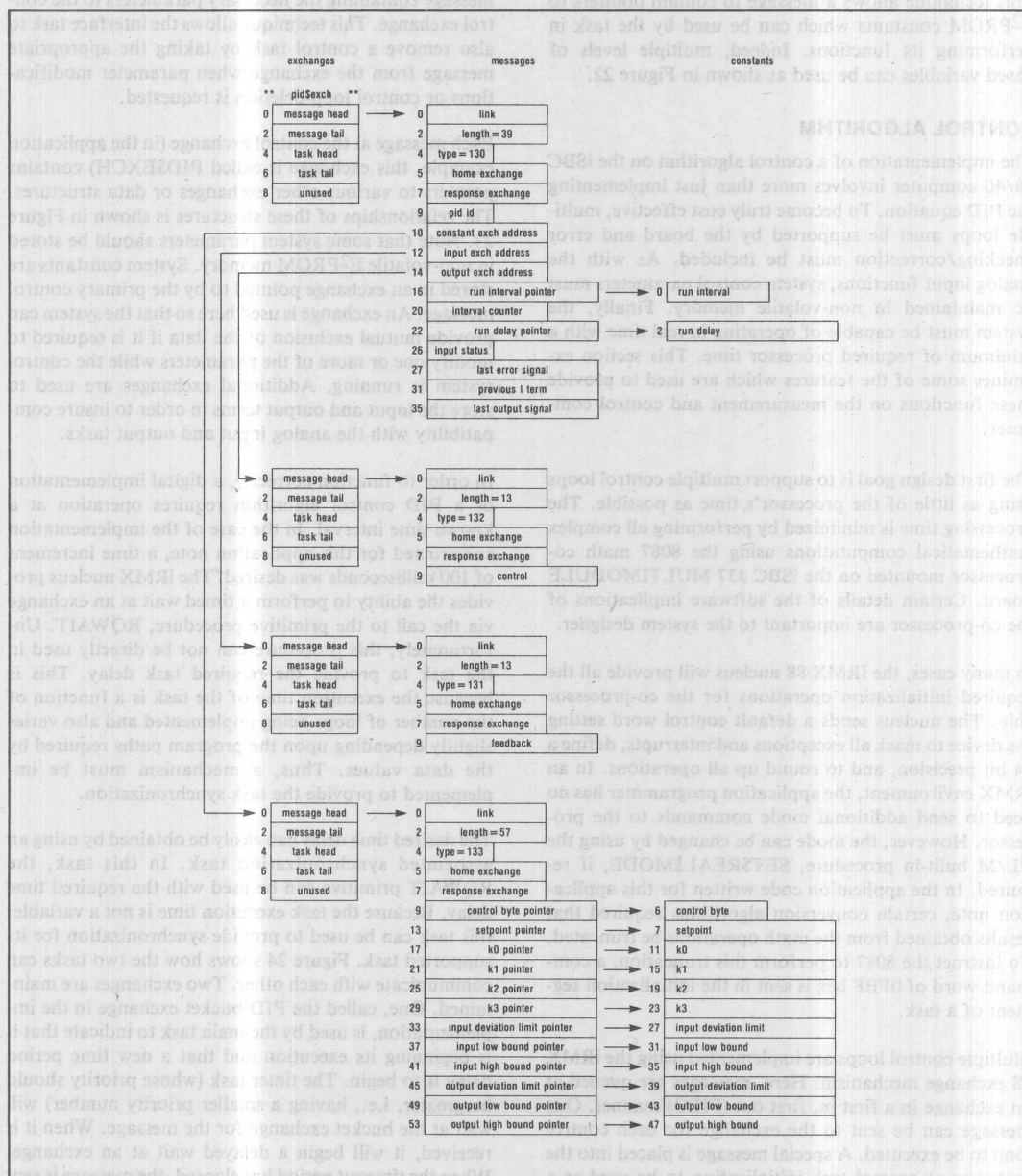


Figure 23. Control Structure Relationships

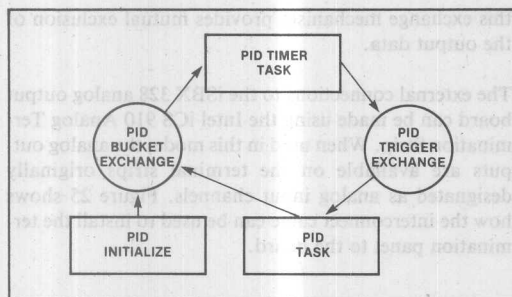


Figure 24. Synchronization Task

Normally, interrupt level 1 is used for this timer; however, any available level may be selected and the iRMX nucleus can be configured to operate correctly by the Interactive Configuration Utility (ICU). A later section of this application note will explain the ICU interaction in more detail. The timed delay function will allow delay increments as small as one millisecond. Each call to the delayed wait function specifies the number of delay increments for which to wait (0 to 65535 increments may be specified in the call). Care should be taken when specifying a delay period of only one unit. The system is not always capable of resolving this delay accurately and an indeterminant delay of from 0 to 1 unit may actually elapse.

ANALOG OUTPUT FUNCTIONS

Once the feedback loop of a control system has been sampled and a control signal generated by the PID control algorithm, a final conversion must be made to provide an output compatible with the system control element. Because such a wide variety of control elements is available, the iSBC 88/40 board was designed to accept the output control circuitry as an expansion option rather than to build unnecessary components and drivers into the board.

In most cases, the control element is driven with an analog signal, either a 4-20 milliamp current signal or a DC voltage. The iSBC 88/40 board is easily interfaced to the analog world using the iSBX 328 Analog Output MULTIMODULE Board. The use of this board is so common with the measurement and control computer that it warrants the time to explain its operation in some detail.

Each iSBX 328 board provides up to eight voltage or current outputs which can drive a wide variety of control devices.

The use of intelligence on the MULTIMODULE expansion board is a key element providing the ability to incorporate eight channels in a very small physical area. The

capabilities of the intelligence allow the board to provide significant enhancements to the basic operational characteristics of the host iSBC 88/40 board. One example is the ability to perform diagnostics of the analog output module upon command from the 8088 processor on the host measurement and control board.

The expanded capabilities bring with them a requirement for some care on the part of the system designer. A fixed programming sequence and handshaking are required to reliably communicate with the expansion board. An analog output driver is easily written which provides the necessary support for analog output signals associated with the control application.

Each time a reset is issued to the iSBX 328 board, it executes a test of its internal stored program to assure data integrity and of its usable RAM to insure that each location can be written to and read from correctly. If either of these tests fail, a bit in the status field will be set to indicate the failure. If the test is performed satisfactorily, the F0 status register (bit 2 of the base port + 2) is set to indicate that the board is ready to receive an initialization command. The initialization command is used to specify the operational mode and number of channels being used.

The operational mode specifies which of four internal programs are to be used to move data between the iSBX interface and the outside world. Each program specifies a unique hardware configuration of the board. Two programs are associated with unipolar operation of the DAC outputs. Program 1 is used when some channels are associated with voltage outputs and some are configured as current outputs. Data directed to a current output will be internally scaled and offset before being sent to the DAC. Data specified as directed to a voltage output is not modified by the program. Program 2 indicates that either all eight channels are used for voltage outputs or that all eight channels are used for current output. Current outputs will be offset by the hardware but no scaling is accomplished. This program 2 mode results in a 10% increase in performance over what is specified in the data sheet of the iSBX 328 board.

Both unipolar programs assume that the data is pure binary formatted with a 0 hex corresponding to a voltage level of 0 volts (or 4 milliamps). A value of 0FFF0 hex will generate a voltage of 4.99 volts (in the current configuration mode, program 1 will result in a 20 milliamp current while program 2 will result in an output of 24 milliamps).

There are also two operational modes which can be used to support a bipolar operation. Program 2 provides a direct hardware support capability for those cases where

all outputs are either configured as entirely voltage or entirely current outputs. No adjustments are made to the data prior to being sent to the DAC. The data format used for both bipolar modes is the offset binary representation of a number. Negative numbers are represented by the values 0 (-32752) to 8000 hex (0). Positive numbers range from 8000 hex (0) to 0FFF0 hex (+32752). Channels defined as current outputs have no legal negative output values.

Finally, program 4 is used to support bipolar operations where the outputs are mixed between current and voltage. The program does not alter data destined to voltage channels but does offset and scale data for channels designated as current outputs.

Once the device has been initialized, subsequent data transfers are all through the data transfer port located at the base address of the board's MULTIMODULE socket. Before each write to the device, the driver software must check the IBF bit (bit 1) of the status to verify that the input buffer is not full. An additional bit is used to specify to the host processor which data byte (high or low) is next to be passed. The low order bits of the low data byte specify which channel the data is for and also what configuration (voltage or current) corresponds to that channel.

Like the analog input task, the application driver for the analog output can be an iRMX 88 task using exchanges and messages for data transfer. In the example implemented for this application note, an exchange, DAC\$EXCH, was dedicated to the control of the task. It contains messages which specify the output port, channel used, and output mode (current or voltage). The task runs at a twenty millisecond time interval and updates each channel as indicated by the control messages. The location of the exchange used to store the output data is also specified by the control message. The use of

this exchange mechanism provides mutual exclusion of the output data.

The external connections to the iSBX 328 analog output board can be made using the Intel iCS 910 Analog Termination Strip. When used in this mode, the analog outputs are available on the terminal strips originally designated as analog input channels. Figure 25 shows how the interconnect cable can be used to install the termination panel to the board.

E²PROM FUNCTIONS

Several references have been made to the advantages gained by using E²PROM 2816 devices on the iSBC 88/40 board for the storage of non-volatile variables. Many configurations mixing E²PROM devices with combinations of EPROM, ROM, and/or byte wide RAM are possible. Support is provided for the installation of one, two, four, or eight (using the iSBC 341 MULTIMODULE EPROM board) 2816 devices. Complete E²PROM write capability is provided on the board. The Intel supplied hardware for this support includes a switching power supply and wave-shaping circuitry. Only minimal user programming overhead is required by the application program.

The on-board wave-shaping circuitry provides a 2816 compatible programming pulse of approximately 16 milliseconds duration. In order to generate this pulse, use is made of the on-board 8253 programmable interval timer. Wirewrap jumper posts are provided to route the timer output to the pulse generator. The gate to the timer is connected using an additional jumper to the memory decode logic to signal a write request to a 2816 device. User software must be provided to program the 8253 for the generation of a 14 millisecond pulse. This code is most easily located in the initialization portion of one of the application tasks associated with writing data into

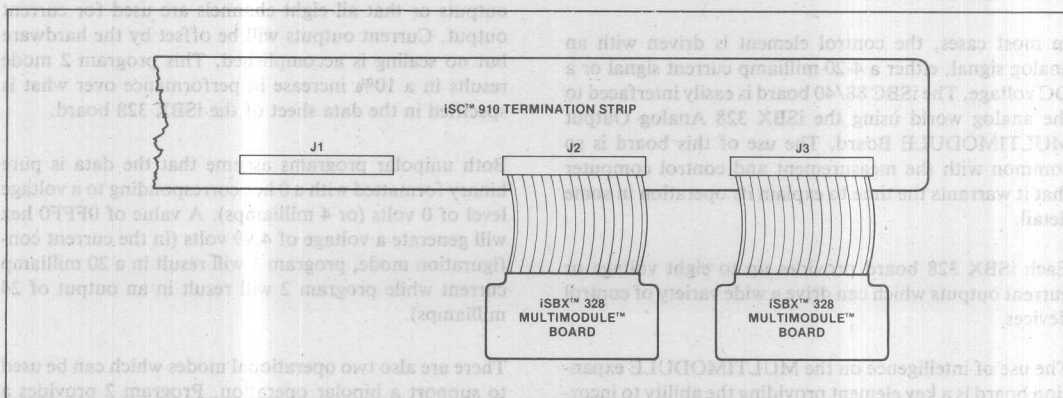


Figure 25. Analog Output Terminations

the devices. Only three lines of PL/M-86 code are required to perform the initialization. The application example includes the code:

```
output(0D6H)=0B2H;
/* timer 2 to mode 2 */
output(0D4H)=000H;
/* most significant byte */
output(0D4H)=038H;
/* least significant byte */
```

The hardware will now generate the appropriate programming pulses to write into the 2816 each time data is written into an address occupied by the device. When the EPROM size is larger than 2K bytes size of the 2816, the system will create a duplicate image of the 2K block as many times as is required to fill the size specified for the EPROM. For example, if 2732A EPROM devices are used and one 2816 is installed at a base location of 0F8000 hex, one image of the E²PROM data will occupy the memory from 0F8000 hex to 0F87FF hex while a second image will be seen from 0F8800 hex to 0F8FFF hex. Reads or writes to either image will access the same data and either may be used.

The user must consider the possibility of system power failures and their impact when designing systems which use the iSBC 88/40 board's E²PROM capabilities. This is especially true in systems whose power supply for the +5 volt source is protected by a crowbar circuit. The on-board switching power supply which generates the high voltage programming pulse operates at very low input voltages and its RC time constant will provide significant voltage levels even if the +5 volt input supply is abruptly removed. The presence of a programming voltage in the absence of a +5 volt supply to a 2816 can cause irreversible damage to the E²PROM chip. The potential for this condition during a write cycle must be considered by the designer. Figure 26 shows a circuit which can be added to the system and connected via the iSBC 88/40 board's P2 connector if desired. The purpose of the circuit is to crowbar the V_{pp} programming voltage to the +5 volt supply if the +5 volt voltage level drops below about 4.5 volts, thus preventing any damage to the 2816.

From a software standpoint, only two items need be given attention during the writing of E²PROM devices on the board. First, before any location can be written in the 2816, the location must first be cleared to an initial value of 0FF hex. Unless this value is already present in the device, two write cycles are required to store new data (the 2816 has a chip erase mode but it is not supported on the iSBC 88/40 board). The second item involves inhibiting interrupts during the write cycle. The programming pulse generation circuitry uses the on-

board timeout circuitry, so the timeout interrupt, if used, must be masked off prior to beginning the write cycle (this implies that the hardware for the timeout acknowledge must be installed to all the circuitry to become a part of the pulse generator). In an iRMX 88 environment using timed waits, the interval timer must also be masked off during the write cycle. If these interrupts are not masked off, the processor can respond to an interrupt and begin modifying its internal registers which point to the memory. This will result in incorrect programming of the E²PROM device. An example of the code which might be used to program a 2816 is shown in Figure 27. The programmer should keep in mind that, during the programming of the 2816, the iAPX 88/10 processor is in a wait state and cannot process any instructions. Thus, for each byte written, approximately 36 milliseconds must elapse before processing can again begin (18 milliseconds for the clearing of the byte and another 18 for the data write). If timed waits are being performed, an error will be introduced into the system. Some critical applications may need to take this into account.

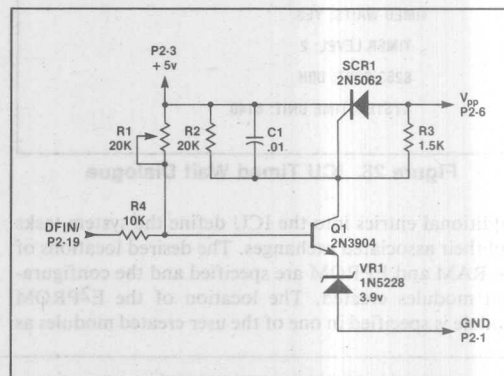


Figure 26. E²PROM Crowbar Protection

```
then do;
366 5  disable;
367 5  call movb(@erase$pattern(0),
           @constants(k).table$pointer, 4);
368 5  constants(k).table$pointer =
           @constants(k).linearization$table(0);
369 5  enable;
370 5  end;
```

Figure 27. E²PROM Programming Example

System Implementation

The application programs for the example described in this application note have been implemented using Ver-

sion 1.1 of the iRMX 88 executive. The use of the Interactive Configuration Utility (ICU88) considerably reduces the effort required to bring a system on line by providing a question and answer session with the programmer. The output of the ICU consists of a system configuration module and a submit file which provides most of the required LINK and LOCATE commands.

In the application, a system time wait increment of 5 milliseconds was chosen. Figure 28 shows the dialogue required to implement the timed wait feature using the board with the output of channel 0 (from the 8253 programmable interval timer) connected to interrupt level 2. Note that the system time unit of 6140 corresponds to a 5 millisecond increment.

```

INTERRUPTS: Y
8259A PORT: COH
8259 INTERVAL: 2
INTERRUPT SERVICE ROUTINE VECTOR BASE: 56

TIMED WAITS: YES
TIMER LEVEL: 2
8253 PORT: DOH
SYSTEM TIME UNIT: 6140
    
```

Figure 28. ICU Timed Wait Dialogue

Additional entries into the ICU define the system tasks and their associated exchanges. The desired locations of the RAM and EPROM are specified and the configuration modules created. The location of the E2PROM module is specified in one of the user created modules as

a public pointer which is initialized with the base address of the device. An example might be:

```

e2prom$module: do;
  declare e2prom$pointer pointer public
  data (0f8000h);
end e2prom$module;
    
```

All references to the data structures in the 2816 are by means of a based variable or structure.

Before executing the submit file for a ROM based system, it is necessary to edit the LOCATE command to include the BOOTSTRAP request. This will assure that the locator places a long jump at the reset vector location when the system is executed out of EPROM. Execution of the LOCATE facility will generate a warning 38 which should be ignored. The corrected LOCATE code is shown in Figure 29.

```

ISIS-II MCS-86 LOCATER, V1.3 INVOKED BY:
LOC86 :F1:ADCNP.LNK TO :F1:ADCNP MAP
PRINT(:F1:ADCNP.MP2)&
BOOTSTRAP ORDER(CLASSES(DATA,STACK,CODE))&
ADDRESSES(CLASSES(CODE(0FC000H), DATA(000400H)))
WARNING 38: SEGMENT WITH MEMORY ATTRIBUTE NOT PLACED
HIGHEST IN MEMORY SEGMENT: MEMORY
    
```

Figure 29. LOCATE Modification

The total control system for the application example can now be assembled using the hardware and software discussed in this note. The same "black box" approach used in hardware designs can be extended to include both software and hardware implementations. Figure 30 shows the complete solution to the control of up to eight agitated heating tanks.

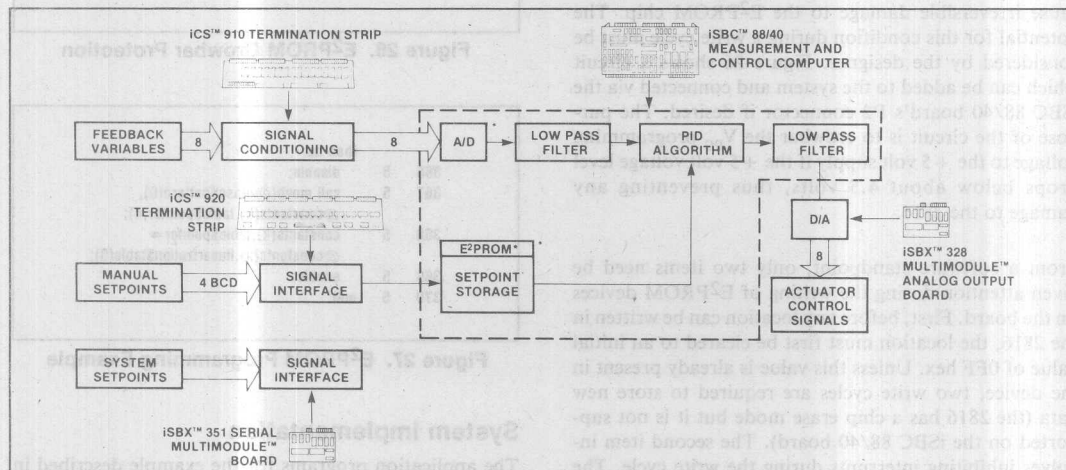


Figure 30. System Implementation

CONCLUSIONS

The purpose of this application note is to illustrate how the Intel iSBC 88/40 Measurement and Control Computer can be used to solve a complex control application. This has been done in Intel's lab and the results obtained from operating the board indicate that the system performance is sufficient to support the operation of eight loops each 100 milliseconds. Observed operation of the analog input and engineering unit conversion task indicated a 4 millisecond per channel execution time. The actual PID code required only 5 milliseconds per loop to execute.

The ease of implementation and fast execution time for multiple complex loops is a result of many Intel product features. For example, the use of the iRMX 88 Real Time Executive provided fast, small, and easy-to-use multitasking real time software for use on the single board computer. The iSBC 337 MULTIMODULE Numeric Data Processor Board enabled the use of high ac-

curacy, easy-to-use floating point calculations without taking excessive execution time. If desired, a fixed point integer math algorithm could have been substituted for the floating point without changing the system performance appreciably. The iSBX 328 Analog Output MULTIMODULE Board provided low cost customization of the base board to support a variety of controllers and actuators.

Finally, the use of the Intel 2816 E²PROM provided the non-volatile storage for system setpoints and constants which is required in a control situation.

Above all, the iSBC 88/40 Measurement and Control Computer provided a platform and execution vehicle for mounting and operation of the various ancillary features. Its iAPX 88/10 processor, memory and MULTIMODULE expansion sockets provided the flexibility to easily customize the board to a particular application environment.

APPENDIX A

easy, easy-to-use floating-point calculations without taking excessive execution time. If desired, a fixed-point integer math algorithm could have been substituted for the floating-point without changing the system performance appreciably. The iSBX 328 Analog Output MULTIMODULE Board provided low cost customization of the base board to support a variety of controllers and actuators.

Finally, the use of the Intel 2816 EPROM provided the non-volatile storage for system setpoints and constants which is required in a control situation.

Above all, the iSBX 88/486 Measurement and Control Computer provided a platform and execution vehicle for mounting and operation of the various ancillary features. Its IAPX 88/486 processor, memory and MULTIMODULE expansion sockets provided the flexibility to easily customize the board to a particular application environment.

CONCLUSIONS

The purpose of this application note is to illustrate how the Intel iSBX 88/486 Measurement and Control Computer can be used to solve a complex control application. This has been done in Intel's lab and the results obtained from operating the board indicate that the system performance is sufficient to support the operation of eight loops each 100 milliseconds. Observed operation of the analog input and converting unit conversion task indicated a 4 millisecond per channel execution time. The actual PID code required only 2 milliseconds per loop to execute.

The ease of implementation and fast execution time for multiple complex loops is a result of many Intel product features. For example, the use of the iRMK 88 Real Time Executive provided fast, small, and easy-to-use multitasking real time software for use on the single board computer. The iSBX 327 MULTIMODULE Input/Output Processor Board enabled the use of high ac-

APPENDIX A

APPLICATION CODE AVAILABILITY

The programs which were used to construct the application example are available from Intel through Insite. Insite, Intel's Software Index and Technology Exchange, is a collection of programs, subroutines, procedures and macros written by users of Intel's 8008, 8080, 8085, 8086, 8088, and 8048 microcomputers. Information on how to join Insite and obtain the source code can be obtained from your local Intel sales office or distributor, or by writing to:

North America

Intel Corporation
User's Library 6-5000
Microcomputer Systems
3065 Bowers Avenue
Santa Clara, California 95051

Europe

Intel International Corp. S.A.
User's Library
Rue du Moulin a Papier 51
Boite 1
B-1160 Brussels, Belgium

Orient

Intel Japan K.K.
User's Library
Flowerhill-Shinmachi, East Bldg.
1-23-9 Shinmachi, Setagaya-ku
Tokyo 154, Japan

APPLICATION CODE AVAILABILITY

Europe
Intel International Corp. 2 A
User's Library
Rue de Moulin à Papier 21
Boite 1
B-1160 Brussels, Belgium

The programs which were used to construct the applica-
tion examples are available from Intel through Intel's
Intel's Software Index and Technology Ex-
change, a collection of programs, subroutines, pro-
cedures, and macros written by users of Intel's 8080,
8085, 8088, 8086, and 8088 microcomputers. In-
formation on how to join Intel's and obtain the source
code can be obtained from your local Intel sales office or
Intel, or by writing to:

China

North America

February 1, 1978

Intel Japan K.K.
User's Library
Flowerhill-Shinbashi, East Bldg.
1-22-2 Shinbashi, Tokyo 100
Tokyo 100, Japan

Intel Corporation
User's Library 6-2000
Microcomputer Systems
3065 Bowyer Avenue
Santa Clara, California 95051

**Reduce your μ C-based system
design time by using single-
board microcomputers**

By George Adams
Electronic Design 3/February 1, 1978

Reduce your μ C-based system design time by using single-board microcomputers. Assembled boards in the SBC-80 series offer stock answers to custom demands.

System designers eager to take advantage of the dramatically increased capabilities of microcomputers have been hindered two ways: Their production volumes have been too low to amortize software and hardware development costs effectively, or hardware subtleties and test requirements have confined them to fully assembled and tested computer subsystems. But now those obstacles are overcome with families of fully assembled and tested microcomputers and system-expansion boards like the Intel SBC-80 series. They are ready-to-use, flexible and inexpensive—prices range from just \$195 to \$825 in unit quantities.

The main members of the SBC-80 family are the 80/04, 80/05, 80/10A, 80/20 and 80/20-4 central-processor boards, with either an 8080A or 8085 microprocessor acting as the master CPU (Table 1). Most of the boards measure 6.75×12 in. and contain the CPU, clock, read/write memory, control ROM, I/O ports, serial communications interface and bus-control logic.

I/O interfacing is an area where design flexibility is essential to meet changing requirements efficiently. The programmable parallel and serial I/O structures of the boards make them versatile enough to do just that. What's more, upgrading system performance is easy thanks to the SBC-80 system bus, the Multibus, which permits modular performance expansion.

The Multibus provides a defined, standard interface between the SBC-80 single-board computers and expansion boards. As many as 16 SBC-80 family boards can simultaneously share the bus.

All in the SBC-80 family

As exemplified by the block diagram of the SBC-80/10A (Fig. 1), the SBC-80 microcomputer system has all that's needed for many applications. The SBC-80/10A is the oldest board in the family and has been widely imitated since it was one of the first "standardized" microcomputers commercially available.

The CPU section of the 80/10A board consists of

the 8080A CPU, the 8224 clock generator and the 8238 system controller. Capable of fetching and executing any of the 8080A's 78 instructions, the CPU section can respond to interrupt requests originating on and off the board. (For more about the 8080A, see "Microprocessor Basics, Part 2," ED No. 10, May 10, 1976, p. 84).

The system-bus interface section includes an assortment of circuits to gate the interrupt and hold requests, the ready signals, and a system-reset signal. Other circuits drive the various control lines. Two 8216s help drive the bidirectional data bus, and six 8226s drive the external system-data and address buses as part of the SBC-80/10A's Multibus interface.

The RAM section of the 80/10A consists of 1024 bytes of static MOS memory. For program storage, up to 8192 bytes of ROM can be mounted on the board in 1024-byte increments by means of a 2708 or 8708 EPROM, an 8308 mask-programmed ROM, or in 2048 byte increments via the 2716 EPROM or 2316 ROM.

A serial interface on the board uses an 8251 programmable universal synchronous/asynchronous receiver/transmitter to provide a serial-data channel. The serial port operates at programmable rates up to 38,400 baud (synchronously) or 19,200 baud (asynchronously) with a choice of character length, number of stop bits, and even, odd or no parity. On-board interfaces provide direct EIA RS-232 or teletypewriter current-loop compatibility.

Two 8255 programmable peripheral interface circuits provide 48 I/O lines for transferring data to or from peripheral devices. Eight already-committed lines have bidirectional drivers and termination networks permanently installed, so that they can be inputs, outputs or bidirectional (jumper-selectable). The other 40 lines are uncommitted. On-board sockets permit drivers and termination networks to be installed, as needed. Since software configures the I/O lines, I/O can be customized for every application.

The 80/10A also responds to a single-level interrupt that can originate from one of many sources, the USART, programmable I/O and two user-designated interrupt-request lines. When an interrupt is recognized, a Restart-7 instruction is generated, and the processor accesses location 38_H to get the starting address of the service routine.

System expansion and support are possible with a

George Adams, Product Line Manager, Single-Chip Microcomputers, Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051.

Note: Multibus, RMX-80, ICE and Inteltec are registered trademarks of Intel Corp.

Reprinted from ELECTRONIC DESIGN/February 1, 1978

Copyright Hayden Publishing Co., Inc. 1978. All rights reserved.

ELECTRONIC DESIGN 3, February 1, 1978

wide variety of alternate-source CPU, memory, and I/O boards (Tables 2 and 3). Up to 65,536 bytes of ROM, PROM or RAM can be accessed by one 80/10A. Expandable backplanes and card cages are also available to support multiboard systems.

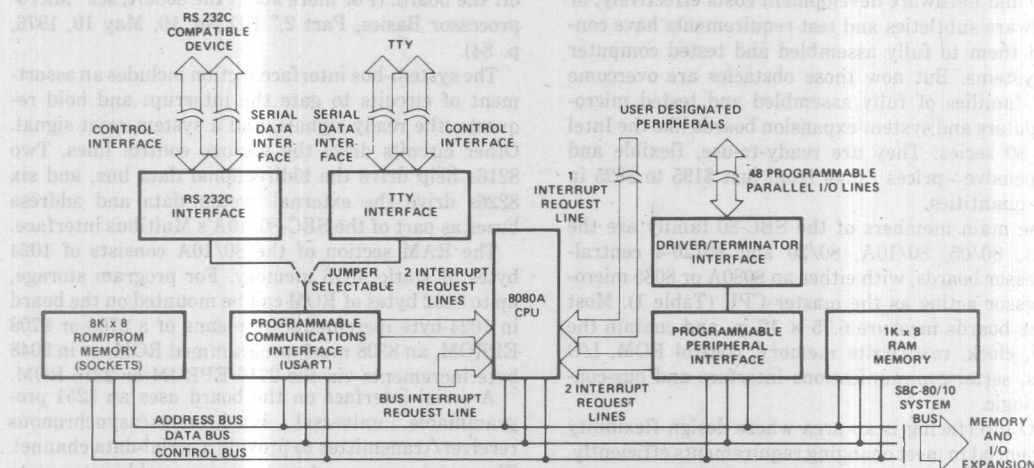
Interfacing starts with the bus

Although the SBC-80/10A is a complete microcomputer system, it can be expanded readily or it can serve as a primary master controller for other microcomputer cards. The 80/10A has five edge connectors, three on the top of the board and two on the backplane, or bottom, side. Two of the "top" connectors, J₁ and J₂, serve as parallel I/O ports, while J₃ is a serial I/O

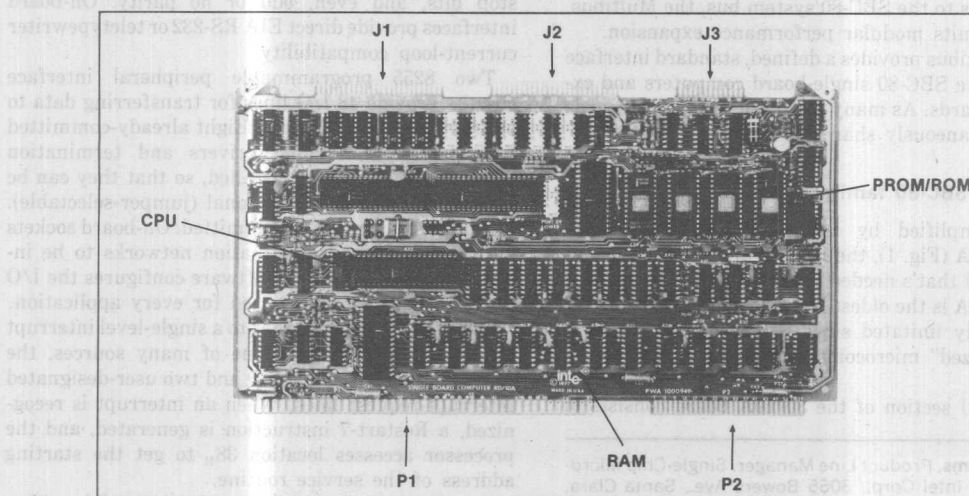
port. All parallel I/O lines on the 50-contact J₁ and J₂ connector areas are paired with an independent signal/ground pin to permit alternate signal/ground wiring when using flat-cable interconnects. Serial port J₃ uses a 26-contact PC-edge connector to provide interfaces for both RS-232 and current-loop devices.

To communicate with other system-compatible boards, the 80/10A uses the 86-pin Multibus (P₁). To provide accessible test points, the 80/10A has a 60-pin edge connector (P₂). The control signals on the Multibus provide the real power and capability in control applications.

Of the 86 pins that make up the Multibus, 24 are assigned to power and ground, 16 to addressing, eight to bidirectional data, and 12 to signal and control

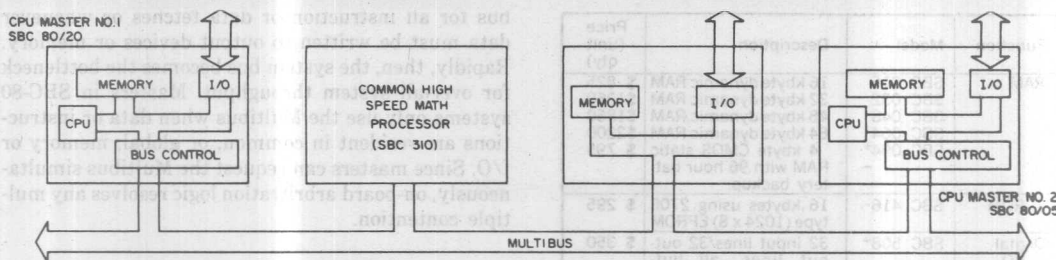


Interrupts originating from the Programmable Communications Interface and Programmable Peripheral Interface are jumper selectable.



1. Based on an 8080A μ P, the 80/10A microcomputer has a straightforward design suitable for general-purpose

computing and control. The board has 48 programmable I/O lines and serial interfaces.



2. The Multibus interface for the SBC-80 CPU boards not only permits simultaneous multiprocessing, but also enables several processors to share the same bus and

peripheral devices. Arbitration logic on the CPU boards decides which board gets on the bus first if several units simultaneously access the bus.

(these 12 are defined in Table 4). The remaining 26 pins are unassigned at this point. Higher capability SBC-80 products, though, are in development. These boards will use many of the unassigned lines (eight unassigned pins are allocated for additional bidirectional data lines). The remaining lines provide multi-level (eight) interrupt lines, various control lines and a multimaster, bus-arbitration control structure (Fig. 2). Address and data lines are three-state, and the interrupt and control lines are open-collector.

Boards using the Multibus have a master-slave relationship: A bus master—such as an SBC 80 CPU board, a DMA controller or a diskette controller—can control the command and address lines. Conversely, slave boards—such as a memory, I/O-expansion or mathematics boards—cannot control the bus.

Arbitration resolves priority disputes

In multimaster systems, the bus-arbitration logic uses the CCLK signal of the bus to provide a timing reference to help satisfy many simultaneous requests for bus control. As a result, different speed masters

can share resources on one bus. Actual transfers on the bus proceed asynchronously with respect to the bus clock. Once bus access is granted, single or multiple read/write transfers can proceed at up to 150 kbytes/s for CPU operations and up to 1 Mbyte/s for DMA operations. The bus has a bandwidth of 5 Mbytes/s so that future performance enhancements may be directly supported.

Both serial and parallel modes of bus-priority resolution are available. In the serial mode, up to three masters can share the system bus, with requests ordered on the basis of bus location. Each master on the bus notifies the next one down in priority when it needs to use the bus, and monitors the bus-request status of the closest higher-priority master. With an external priority network, up to 16 masters can share the bus.

The dual-bus nature of the Multibus permits each processor-based master within the system to retain its own local memory and I/O, which it uses for most operations. Such local operations occur entirely on the individual board and don't require the system bus.

In contrast to the dual bus architecture, all masters

Table 1. Comparison of SBC-80 CPUs

	SBC 80/04	SBC 80/05	SBC 80/10A	SBC 80/20	SBC 80/20-4
CPU	8085	8085	8080A	8080A	8080A
EPROM capacity (bytes)					
(with 2716)	4096	4096	8192	8192	8192
(with 2708)	2048	2048	4096	4096	4096
RAM (bytes)	256	512	1024	2048	4096
Programmable parallel I/O lines	22	22	48	48	48
Serial I/O capability	RS232C SID/SOD ^{1, 2}	RS232C SID/SOD ^{1, 2}	RS232C/TTY USART	RS232C ² USART	RS232C ² USART
Timers	1	1	0	2	2
Interrupt levels	4	4	1	8	8
Multibus interface	None	Multi-master	Single-master	Multi-master	Multi-master
Price (unit quantity)	\$195	\$350	\$495	\$735	\$825

Notes: ¹Provided by 8085 CPU SID and SOD serial I/O lines. ²Optional SBC 530 TTY interface is available.

Table 2. Additional SBC support boards

Function	Model	Description	Price (unit qty)
RAM	SBC 016	16 kbyte dynamic RAM	\$ 825
	SBC 032	32 kbyte dynamic RAM	\$1360
	SBC 048	48 kbyte dynamic RAM	\$1860
	SBC 064	64 kbyte dynamic RAM	\$2200
	SBC 094*	4 kbyte CMOS static RAM with 96 hour battery backup.	\$ 795
EPROM	SBC 416	16 kbytes using 2708 type (1024 x 8) EPROM	\$ 295
Digital I/O	SBC 508*	32 input lines/32 output lines, all buffered/terminated	\$ 350
	SBC 517	48 programmable parallel lines with full buffering/termination options, full RS232C port, 1 ms real-time clock, and 8-line interrupt control	\$ 400
	SBC 519*	72 programmable parallel lines with full buffering/termination options, real-time clock (interval is jumper selectable to 0.5, 1, 2, or 4 ms), and 8-level programmable interrupt control.	\$ 395
Communications	SBC 534	Four programmable synchronous/asynchronous serial ports, each with: programmable baud rates, programmable data formats, programmable interrupt control, 16 RS232C buffered programmable parallel I/O lines configured as a Bell Model 801 automatic calling unit interface. Two programmable 16-bit interval timers (usable as real-time clocks), and software selectable loop-back of serial ports for diagnostic use.	\$ 650
	SBC 556*	48 optically isolated lines; 24 input 16 output, and 8 programmable (in/out), 8-level programmable interrupt control, and 1 ms real-time clock.	\$ 395
Analog I/O	SBC 711*	16/8 (single-ended/differential) 12-bit a/d channels; user expandable on-board to 32/16 channels	\$ 895
	SBC 724*	Four 12-bit d/a channels	\$ 750
	SBC 732*	Combination analog I/O; same a/d capability as SBC 711 plus 2 d/a channels	\$1125
Combination memory and I/O	SBC 104	8 kbytes capacity (sockets) using 2716 (2 k x 8) EPROM or 4 k using 2708, 4 kbytes dynamic RAM, 48 programmable parallel I/O lines, with full buffering/termination, as options. RS-232C port, a 1 ms real-time clock, and an eight-line interrupt control	\$ 715

*Requires +5 V only.

in multimaster/single-bus systems use the common bus for all instruction or data fetches or whenever data must be written to output devices or memory. Rapidly, then, the system bus becomes the bottleneck for over-all system throughput. Masters in SBC-80 systems only use the Multibus when data or instructions are resident in common, or global, memory or I/O. Since masters can request the Multibus simultaneously, on-board arbitration logic resolves any multiple contention.

Examine board performance

A look at the entire family of SBC-80 microcomputers reveals varied levels of performance. All five boards are inexpensive, but the most inexpensive is the 80/04, which costs \$99 in 100-unit quantities, and is intended for stand-alone applications. To get the cost down, the board was designed to use the 8085 CPU and the 8155 RAM, timer and I/O circuit.

The 80/04 contains an 8085 CPU, 256 bytes of RAM, space for up to 4 kbytes of EPROM (two 2716 EPROMs, or two 2708 EPROMs), 22 programmable parallel I/O lines with sockets for buffer and termination options, a 14-bit programmable timer/event counter, and provision for an RS-232-C serial port using the 8085 SID/SOD serial interface. The board can also house an on-board +5-V regulator, so an unregulated voltage can be connected.

The next step up, the 80/05, has the same architecture and connector types and pinouts as the 80/04. Direct software compatibility is achieved with the same CPU along with the same RAM, ROM, I/O, and timer addressing. However, the 80/05 contains twice as much RAM as the 80/04. And since the 80/05 has the full Multibus multimaster interface, 80/05-based systems can be expanded with any of the Multibus-compatible boards from Intel or other suppliers.

The SBC-80/10A comes next. It provides more on-board memory and I/O for systems requiring expanded on-board resources. Based on the 8080A CPU, the board contains 1 kbyte of RAM, up to 8 kbytes of EPROM/ROM, 48 programmable parallel I/O lines, a full USART serial port with RS-232-C and tele-

	SBC 108	Same as SBC 104, except has 8 kbytes of dynamic RAM	\$ 815
	SBC 116	Same as SBC 104, except has 16 kbytes of dynamic RAM	\$ 985
High-speed math	SBC 310*	High speed mathematics processor including floating-point capability (32 bit).	\$ 595
Peripheral control	SBC 201	Dual single-density diskette controller	\$ 995
	SBC 202	Quad double-density diskette controller	\$1290
DMA control	SBC 501	DMA controller, up to 1 MHz transfer rates	\$ 450

typewriter interfaces, and a full Multibus interface (but only single-master capability; the board has no multimaster capability). Intended for single-CPU systems with only one other Multibus peripheral controller, the 80/10A can interface with such as the SBC-201 or SBC-202 single and double-density diskette controllers, or the SBC-501 DMA controller.

System designers requiring the same on-board I/O capability as the SBC-80/10A but with more RAM, more efficient real-time capability, and full multimaster Multibus control can go further up the ladder to the SBC-80/20 or SBC-80/20-4. These boards differ only in that the 80/20 contains 2 kbytes of RAM and the 80/20-4 contains 4 kbytes. Both boards can hold up to 8192 bytes of ROM or EPROM, handle up

to eight levels of prioritized interrupt, and share the Multibus in the multimaster mode. Either board has two programmable interval timers/event counters. Auxiliary power buses and memory-protect control logic on the board permit battery backup of the RAM.

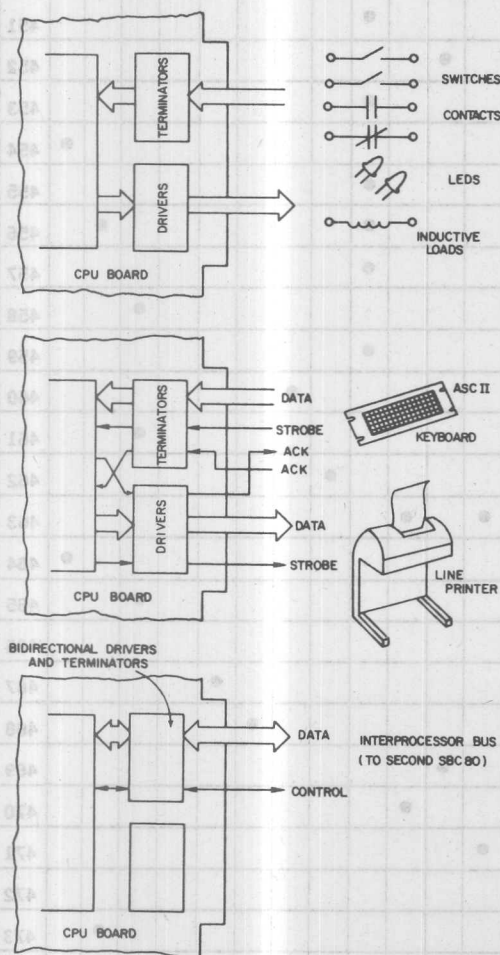
Take advantage of interrupts and timers

Real-time applications frequently require that high-priority programs operate on the basis of external events, time-of-day, or elapsed time without impacting current background processing. These multiprogramming requirements are supported in the 80/20 and 80/20-4 by an eight-level programmable interrupt controller (PIC) and two programmable interval timer/event counters. The priority level of any event generating an interrupt request is assigned through jumper selection and the priority algorithm chosen by system software.

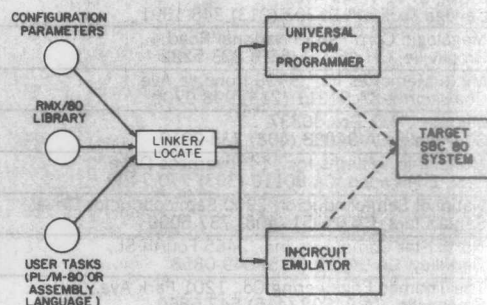
Any combination of interrupt levels may be masked by storing a single byte in the interrupt-mask register contained by the PIC, whose four software-selectable priority algorithms are described in Table 5. The PIC generates a unique memory address for each interrupt level. These addresses are equally spaced at intervals of 4 or 8 bytes (software-selectable). The resulting 32 or 64-byte block may begin at any 32 or 64-byte boundary in the 65,536-byte memory space. A single 8080A jump instruction at each of these addresses then provides linkage to locate each interrupt service routine independently anywhere in memory.

The two programmable timers may be used to generate real-time clocks by requesting periodic interrupts through the PIC, so that the CPU is free to handle numerous other system-timing and control functions. The outputs and gate/trigger inputs of the timer/counters can be routed via jumpers to the PIC, the I/O driver/terminators, or the programmable parallel I/O.

Seven software-selectable timing/counting functions are available. Either timer may be set to act as a rate generator (divide-by-N counter), a square-wave



3. Programmable I/O lines from the SBC-80 parallel interfaces can be set so that they are individually programmable as inputs or outputs (a), byte-programmable as inputs or outputs with handshaking (b), or bidirectional on a byte-programmable basis (c).



4. By using the RMX-80 executive and the library of often-used subroutines, program development can be simplified since the subroutines are modular and can be linked together, then checked out in a system prototype.

Table 3. Non-Intel SBC-compatible boards

Manufacturer	Multibus CPU boards	RAM boards	Core memory boards	PROM/ROM boards	Analog boards	AC power control boards	IEEE-488 bus	Tape interface	Floppy-disc controller	Communications interface	Video boards	Math processors	Breadboards	Circle no.
ADAC Corp., 118 Cummings Park, Woburn, MA 01801.(617) 935-6668					•									451
Ampex, Memory Products Div., 200 N. Nash St., El Segundo, CA 90245.(213) 640-0150			•											452
Analog Devices, Route 1 Industrial Park, P.O. Box 280, Norwood, MA 02062.(617) 329-4700					•									453
Augat Inc., 33 Perry Ave., P.O. Box 779, Attleboro, MA 02703.(617) 222-2202													•	454
Burr-Brown, International Airport Industrial Park, P.O. Box 11400, Tucson, AZ 85734.(602) 294-1431					•									455
Cybernetic Microsystems, 2460 Embarcadero Way, Palo Alto, CA 94303.(415) 321-0410					•							•		456
Data Translation Inc., 23 Strathmore Road, Natick, MA 01760.(617) 655-5300					•									457
Datacube Corp., 25 Industrial Park, Chelmsford, MA 01824.(617) 256-2555											•			458
Datel Systems Inc., 1020 Turnpike St., Building S, Canton, MA 02021.(617) 828-8000					•									459
Digidata Corp., 8580 Dorsey Run Road, Jessup, MD 20794.(301) 498-0200							•							460
EDAC Corp., 1417 San Antonio Ave., Alameda, CA 94501.(415) 521-6600											•			461
Electronic Engineering & Prod. Services, TE. #2, Louisville, TN 37777.(615) 984-9640						•								462
Electronic Solutions, 7969 Engineer Rd., San Diego, CA 92111.(714) 292-0242	•	•	•											463
Garry Mfg. Co., 1010 Jersey Ave., New Brunswick, NJ 08902.(201) 545-2424													•	464
Hal Communications Corp., Box 365B, 807 E. Green St., Urbana, IL 61801.(217) 367-7373											•			465
Iasis, 815 W. Maude Ave., Sunnyvale, CA 94086.(408) 732-5700	•													466
ICOM, 6741 Variel Ave., Canoga Park, CA 91303.(213) 348-1391									•					467
Megalogic Corp., 9650 National Road, Brookville, OH 45309.(513) 833-5222								•						468
Micro Memories Inc., 9438 Irondale Ave., Chatsworth, CA 91311.(213) 998-0700			•											469
Microtec, P.O. Box 60337, Sunnyvale, CA 94088.(408) 733-2919				•										470
Monolithic Systems Inc., 14 Inverness Drive, East, Englewood, CA 80110.(303) 770-7400	•	•												471
National Semiconductor, 2900 Semiconductor Drive, Santa Clara, CA 95051.(408) 737-5000	•													472
North Star Computers Inc., 2465 Fourth St., Berkeley, CA 94710.(415) 549-0858													•	473
The Thomas Engineering Co., 1201 Park Ave., Emeryville, CA 94608.(415) 547-5860										•				474
Vector Electronic Products, 12460 Gladstone Ave., Sylmar, CA 91342.(213) 365-9661													•	475
Zia Tech., 10762 La Roda Drive, Cupertino, CA 95015.(408) 996-7082							•							476

generator, a programmable retriggerable one-shot, or software or hardware-triggered strobe. One of the timers can be jumper-selected as an event counter, and either can generate an interrupt after a specified interval or after a specified number of events.

The programmability of each on-board timer allows timing intervals from approximately 2 μ s to over 60 ms. But the two timers may be cascaded to provide intervals greater than 1.1 hour, in 1.86 μ s increments. In the event counter mode, external event rates up to 1.1 MHz may be counted.

Flexible I/O, a must for any system

All SBC-80 microcomputers provide 22 or 48 programmable parallel I/O lines that, grouped as 8-bit ports, are fully programmable to allow enough flexibility to handle any changes in system interfacing. Programmability is permitted through data direction, control mode, interrupt handling, and buffer/termination. The I/O configuration for a specific application is selected through software initialization of the parallel I/O control logic, jumper selection of control/interrupt line routing, and the particular buffer and termination devices chosen.

Fig. 3 illustrates the basic modes of operation that may be selected by software to meet application requirements. Mode 0 is used for slow-to-medium-speed interfacing where immediate handshake response or interrupt generation is not needed. This mode is extremely useful for interfacing to inputs such as switches or outputs such as LED indicators or numeric displays.

Mode 1 provides handshaking lines required for many medium to high-speed peripherals. A typical output function could be a line printer; an input device could be an encoded keyboard or paper tape reader.

In addition, the 80/10A and 80/20 have Mode 2, a bidirectional data/control structure. This interface may provide, for example, a communication link between parallel processors.

The SBC-80 I/O structure also permits multiple options for output buffering and input termination. TTL drivers with 16 to 48 mA of drive can be used, and input lines may be terminated to minimize the impact of noise and cable disconnects. Any of the TTL drivers (four outputs) or input terminators (for inputs) listed in Table 6 may be inserted into sockets to provide proper buffering or termination.

Like the design flexibility of the SBC-80 parallel I/O structure, the serial I/O structure allows interface characteristics to be revised rapidly through software, jumper, and buffer changes. Besides the SBC-80/10A, the 80/20 and 80/20-4 contain the USART serial channel. These boards provide RS-232 interfaces, but the SBC-80/10A also has a teletypewriter current-loop interface. Synchronous/asynchronous mode, data format, control-character format, and parity are all under program control. So is baud rate on the 80/20 and 80/20-4. Baud rate is jumper-selectable on the

Table 4. Multibus control signals

AACK	Advance-acknowledge signal, used in 8080A-based systems. It is sent to the SBC-80 board by a memory bank in response to a memory-read command, allowing the memory to complete the access without requiring the CPU to wait.
BCLK	Bus clock, used to synchronize bus-control circuits on all master boards. It has a period of 101.725 ns (9.8304 MHz) and a 30 to 70% duty cycle. The signal may be slowed, stopped or single-stepped.
BPRN	Bus-priority-input signal, used to indicate to the master that a higher-priority master board wants to use the system bus. When brought high, the signal suspends processing activity and places line drivers of the master in a standby mode.
BUSY	Bus-busy signal, a bidirectional control line that allows control and monitoring of the Multibus in multimaster systems. As an output from a bus master, BUSY indicates the bus is being used by the board. It prevents all other master boards from gaining control of the bus. Each master monitors BUSY as an input to determine current Multibus usage status.
CCLK	Constant clock, used to provide a 9.8304-MHz clock signal for optional memory and I/O expansion boards. CCLK coincides with BCLK and has a period of 101.725 ns and a 30 to 70% duty cycle.
INIT	Initialize signal, used to reset the entire system to a known internal state.
INTR1	Interrupt input, used to interrupt the processor via an externally generated interrupt request.
IORC	I/O-read command, a signal generated by the master to indicate that the address of an input port has been placed on the system-address bus and that the data at that input port are to be read and placed on the system-data bus.
IOWC	I/O-write command, a signal generated by the master to indicate that the address of an output port has been placed on the system-address bus and that the contents of the system-data bus are to be output to the addressed port.
MRDC	Memory-read command, a signal generated by the master that indicates that the address of a memory location has been placed on the system-address bus. It specifies that the contents of the addressed location are to be read and placed on the system-data bus.
MWTC	Memory-write command, a signal generated by the master to indicate that the address of a memory location has been placed on the system-address bus. It causes information on the data bus to be written into the addressed memory location.
XACK	Transfer-acknowledge signal, an input signal to the master board from an external memory location or I/O port to indicate that a specified read or write operation has been completed.

80/10A CPU board.

The synchronous and asynchronous nature of the serial interface makes it compatible with virtually every standard serial data-transmission technique used today (including IBM's Bi-Sync). This allows multiple SBC-80 boards to be interconnected as a distributed-processing network. The resulting task segregation or redundancy (or both) significantly improves both system performance and reliability.

Two jumper-selectable interrupt requests may be generated automatically by the serial interface. One occurs when a newly received character is ready to be loaded into the CPU (receive-channel buffer is full). The other occurs when new data are ready to be transmitted to the remote device (transmit-data buffer is empty).

Both the SBC-80/04 and 80/05 provide serial I/O capability through the serial input data (SID) and serial output data (SOD) functions of the 8085 CPU. These functions are controlled by software executing the 8085 read-interrupt mask (RIM) and set-interrupt mask (SIM) instructions.

For systems requiring many serial channels, the SBC-534 communications-expansion board provides four USART channels with RS-232-C and optically isolated current-loop interfaces, programmable interrupt, timing, baud-rate control, and a Bell 801 Auto-Call unit interface.

Expand the system via the Multibus

The SBC-80 family is gaining not only in popularity but in support for its Multibus as more and more companies offer SBC-compatible boards. Intel now provides high-speed mathematics, RAM, EPROM, mass storage, digital I/O, combination memory and I/O, serial communications, and analog-I/O expansion boards.

For applications requiring fast, high-precision number crunching, the SBC-310 math unit acts as an intelligent slave to perform floating-point and fixed-point mathematics. A processor uses the 310 by passing parameters to it along with a command byte to select the desired operation from the SBC-310's 14 instructions. The repertoire includes 32-bit floating-point (single-precision) addition, subtraction, multiplication, division, squaring, square root, comparisons, and tests; 16-bit fixed-point multiply, subtract, extended divide, and extended compare; and conversion from fixed to floating point or vice versa.

A completed operation may be signaled either by the math unit via an interrupt or by the host processor's polling the "operation complete" flag in the unit's status register. The result may be retrieved at this point or left in the 310's accumulator for further use.

In addition, the 310 provides control circuitry so that it may be treated as a "shared resource" among several CPU boards.

Two diskette options are available for mass storage.

Table 5. Programmable interrupt modes, SBC-80/20-4

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Autorotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence on this assignment.
Polled	System software examines priority-encoded system interrupt status via interrupt status register.

Table 6. Line drivers and terminators

Line drivers		
Driver	Characteristic	Sink current (mA)
7438	I,OC	48
7437	I	48
7432	NI	16
7426	I,OC	16
7409	NI,OC	16
7408	NI	16
7403	I,OC	16
7400	I	16

Note: I = inverting; NI = noninverting; OC = open collector

The SBC-201 diskette controller provides full control for one or two single-density diskette drives and acts as a programmable slave to masters on the Multibus. All diskette information is stored in the IBM soft-sectored format. For systems requiring greater storage capacity, the SBC-202 provides full control for up to four double-density diskette drives. Thus, 2 Mbytes of mass storage may be added to SBC-80-based systems for each SBC-202 plugged into the bus.

Digital I/O may be expanded using any of three Intel boards. The SBC-519 provides 72 programmable parallel I/O lines as well as interrupt handling and a real-time clock.

The 519's clock can interrupt the appropriate CPU periodically so that the CPU can monitor system-I/O status. High-speed I/O events can gain the CPU's attention via interrupts. The SBC-517 combination I/O board and the SBC-104, 108 and 116 combination memory and I/O boards offer 48 programmable parallel lines, a full RS-232 USART serial channel, interrupt handling and a 16-ms real-time clock. The

Table 7. RMX-80 routine library

RMX/80 module	Function
Nucleus (executive)	Provides basic capabilities (concurrency, priority, and synchronization/communication) found in all real-time systems.
Terminal handler	Provides real-time asynchronous I/O between an operator's terminal and tasks running under the RMX/80 executive, includes a line-edit feature similar to that of ISIS-II (supervisory system on the Inteltec development system) and type-ahead facility.
Diskette file systems	Diskette driver and file management capabilities, allows user to load tasks into the system and to create, access, and delete files in a real-time environment without disrupting normal processing. File formats compatible with ISIS-II for both single and double-density systems.
Free space manager	Maintains a pool of free RAM and allocates memory out of the pool upon request from a task; reclaims memory areas when no longer needed.
Debugger	Specifically designed for debugging software running under the RMX/80 executive; used by linking it to an application program or task. Thus, it can be run directly from the single-board computer's memory.
Math handler	Provides full control and communication for SBC 310 math board for high-speed fixed and floating-point math functions.
Analog interface handler	Provides real-time control for SBC 711, 724, and 732 analog I/O expansion boards.

104, 108 and 116 also hold up to 8 kbytes of EPROM, along with 4, 8 or 16 kbytes of RAM, respectively.

For systems geared to especially noisy environments, the SBC-556 provides 48 optically isolated I/O lines, which are configured as 24 input lines, 16 output lines, and eight programmable-I/O lines. The user fixes the optical-isolation characteristics according to his exact system requirements by installing the optoisolators and current-limiting resistors of his choice into the board sockets. Input voltages up to 48 V, output lines up to 30 V and currents up to 60 mA may be interfaced.

Of course, many more RAM, ROM, communications and interface options are available. But for systems to come together quickly during development, there must be some standardized operating software to provide some of the most fundamental system routines.

System software: the glue that binds

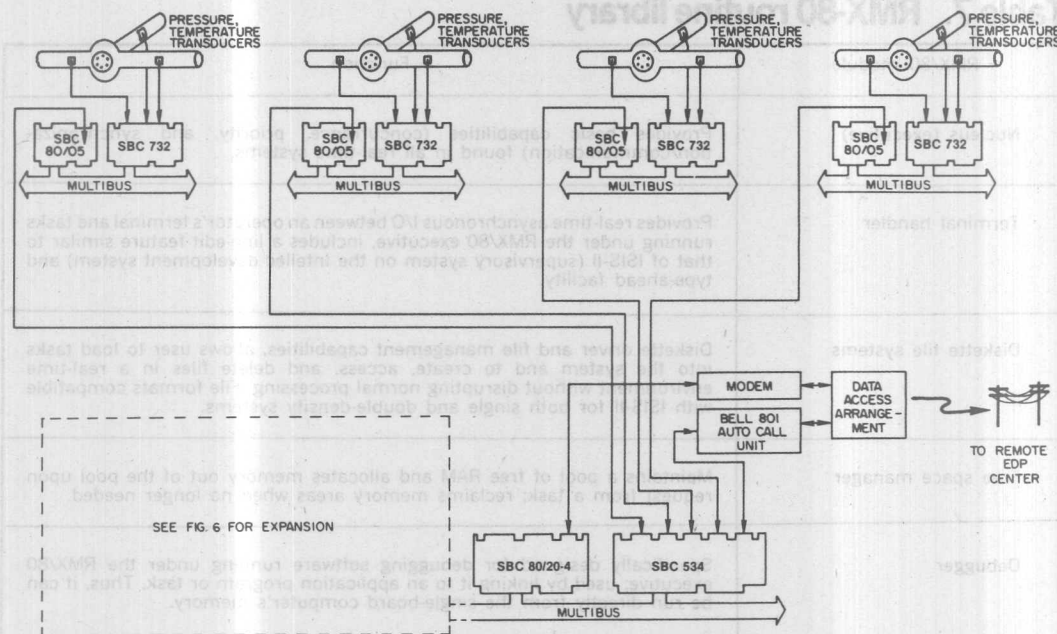
Where the Multibus provides a standard hardware structure, RMX-80, a real-time multitasking executive supplies a modular software framework. With RMX-80, routines don't have to be developed for task synchronization, priority resolution and peripheral control (printers, terminals, diskettes, etc.). Versions

are available for the SBC-80/20, 80/20-4 and 80/10A CPU boards.

Critical projects can be completed rapidly because RMX-80 provides major portions of the software requirements for many real-time systems. For example, the diskette file-extension software of the RMX-80 program may be linked into the system software. Thus, users can immediately have a diskette file structure with facilities to open and close files, create and delete files, read or write files sequentially or randomly (read function may be used for initial program load, if desired), or allocate file storage dynamically on single or double-density diskettes.

The compactness of RMX-80—the entire executive resides in 2 kbytes of ROM—reduces memory requirements and eliminates the need for bootstrap-program loading. All RMX-80 operations are based on individual tasks. A task is a program with unique data and stack that operates asynchronously with other such programs in the system.

Basically, the RMX-80 is a library of "standard" routines (Table 7), such as an analog-interface handler and a terminal handler. Fig. 4 illustrates how to develop software by selecting appropriate RMX-80 modules, then locating and linking them with particular software tasks on an Inteltec microcomputer development system. In addition, a debugger module



5. This possible SBC-80 system configuration uses four SBC-80/05s to monitor and control pipeline parameters

and feed data back to a master controller, an SBC-80/20-4. The master controller sends data back to a host system.

in the RMX-80 speeds real-time system development.

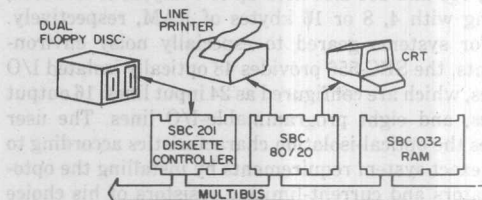
The executive accesses system resources according to task priority, intertask communication, interrupt-driven control for standard devices, real-time clock control, interrupt handling, and other optional functions. In all, there are 255 separate task-priority levels, and since multiple tasks may share the same level, the actual number of tasks is limited only by memory size.

Develop programs with the Intellec

The Intellec and its ICE-80 and ICE-85 in-circuit emulators help minimize the time required to develop software and hardware. Standard Intellec stand-alone software includes resident macroassemblers for the 8080A and 8085 CPUs, a text editor, and a system monitor/debugger. As a result, programs can be assembled, loaded, edited, executed, and debugged.

ICE diagnostics can significantly reduce program development and debug time. Break points may be set on user-specified memory-read or write operations, I/O read or write operations, or user-defined extension parameters. Programs can be single-stepped to check operating conditions and performance.

PL/M-80 is the high-level systems-programming language. The optional Intellec-resident PL/M compiler provides the ability to program in this natural, algorithmic language, so there is no need to manage register usage or to allocate memory. PL/M programs



6. Expanding the pipeline monitor/controller system is as simple as plugging more cards into the Multibus and altering the software. By adding another SBC-80/20 to the master controller, some local processing can be done and a local CRT and high-speed printer can be added as well as RAM and diskette-memory space.

may be linked to assembly-language programs to hasten product development further.

A relocatable macroassembler residing on the Intellec translates symbolic assembly language into 8080 or 8085 machine code and permits the use of relocatable and linkable object code. With full macro capability, similar sections of code needn't be written over and over.

Intellec options include a diskette operating system, ISIS-II, with diskette controller, single or dual diskette

facilities for producing and handling relocatable code, including a relocating macroassembler, relocating loader and a linker to help link separately compiled or assembled programs.

Apply the SBC boards to real use

To get an idea of the SBC 80 family's capabilities, examine the application shown in Fig. 5. In this case, a remote control/monitoring section of a pipeline supervisory control system grows with increasing requirements for additional local throughput and processing capability.

Four SBC-80/05s act as remote pipeline monitors/controllers. Each unit monitors various contact closures (limit switches, relays, etc.) and a hex keypad, with a subset of its own I/O lines programmed as inputs. Contact debounce is performed in software. Other digital I/O lines on each SBC-80/05 act as output lines to drive a numeric display and various control relay coils.

Analog-control lines are interfaced with an SBC-732 combination analog-I/O board. Transducers indicating temperature and pressure drive analog inputs, and analog outputs drive valves. Flow rate is determined in software by manipulating differential pressure data available from pressure transducers.

The four 80/05s are linked serially to a remote

communications expansion board provides four RS-232-C serial channels, each interfacing directly with one of the four 80/05-based pipeline monitor/controllers. The 80/20-4 periodically queries each monitor to determine its current status. The concentrator also relays control commands from a host computer controlling the entire pipeline. The 80/20-4's own RS-232-C serial channel provides the interface for this high-speed synchronous link to the host CPU.

The 80/20-4 can contact the host CPU with the Bell 801 automatic calling-unit interface on the SBC-534. The synchronization and control of communication between the four 80/05s and the host are handled by RMX-80 on the 80/20-4.

The 80/20-4 system can be expanded to provide local processing capability, as shown in Fig. 6. Here, another 80/20 is added as a second master on the Multibus to provide control for a local CRT and high-speed printer, and to provide local processing capability.

An additional 32 kbytes of RAM are furnished by an SBC-032 RAM-expansion board. A third master, an SBC-202 dual-density diskette controller, can also be added to the Multibus, along with two double-density diskette drives. Communication between the two 80/20s is handled via user-written intermaster message tasks.■

communications expansion board provides four RS-232-C serial channels, each interfacing directly with one of the four 80V05-based pipeline monitor/controllers. The 80V20-4 periodically queries each monitor to determine its current status. The host computer controlling the entire pipeline. The 80V20-4's own RS-232-C serial channel provides the interface for this high-speed synchronous link to the host CPU.

The 80V20-4 can contact the host CPU with the Bell 801 automatic call-in-unit interface on the SBC-234. The synchronization and control of communication between the four 80V05s and the host are handled by

The 80V20-4 system can be expanded to provide local processing capability, as shown in Fig. 8. Here, another 80V20 is added as a second master on the Multi-bus to provide control for a local CRT and high-speed printer, and to provide local processing capability.

An additional 32 Kbytes of RAM are furnished by an SBC-082 RAM-expansion board. A third master, an SBC-202 dual-density diskette controller, can also be added to the Multi-bus along with two double-density diskette drives. Communication between the two 80V05s is handled via user-written intermediate message tasks.

drives and 1815-II software. 1815-II provides all facilities for predicting and handling relocatable code, including a relocating macroassembler, relocating loader, and a linker to help link separately compiled or assembled programs.

Apply the SBC boards to test use.

To get an idea of the SBC 80 family's capabilities, examine the application shown in Fig. 8. In this case, a remote control/monitoring section of a pipeline supervisory control system grows with increasing requirements for additional local throughput and processing capability.

April, 1978

monitor/controllers. Each unit monitors various control elements (limit switches, relays, etc.) and a hex keypad, with a subset of its own I/O lines programmed in software. Contact closures is performed in software. Other digital I/O lines on each SBC-80V05 act as output lines to drive a numeric display and various control relay coils.

Analog control lines are interfaced with an SBC-125 communication analog I/O board. Transducers indicate temperature and pressure drive analog inputs, and analog outputs drive valves. Flow rate is determined in software by manipulating differential pressure data available from pressure transducers.

The four 80V05s are linked serially to a remote

Design Motivations for Multiple Processor Microcomputer Systems

George Adams and Thomas Rolander
Microcomputer Applications

DESIGN MOTIVATIONS FOR MULTIPLE PROCESSOR MICROCOMPUTER SYSTEMS

Design decision factors involved in developing multiple processor microcomputer systems include means of minimizing contention for system bus utilization. System applications detail the appropriate hardware and software considerations as related to single-board computers in a multimaster bus structure

George Adams and Thomas Rolander

Intel Corporation, Santa Clara, California

Large-scale integrated circuit technology has reduced the cost of central processors to such a low level that the previously avoided concept of applying multiple processors to meet system performance requirements has now become an attractive and viable alternative. Several key benefits accrue from such an approach. In addition to enhanced system performance (throughput), improved system reliability, and improved system realtime response, modular system expansion capabilities may be realized. Although designing such systems "from scratch" with microprocessor component families can be a complex system design task with many subtle pitfalls which can inhibit efficient system operation, the advent of second generation single-board computers, such as the Intel® SBC 80/05 and 80/20, has allowed multiple processor microcomputer systems to become off-the-shelf products.

Motivation and Design Concepts

Discussion of the benefits of multiple processor structures in system applications will provide an understanding of the motivation for this implementation approach in system design. A primary objective addressed through

multiple processor approaches is enhanced system performance and throughput. Enhanced performance is achieved through partitioning of overall system functions into tasks that each of several processors can handle individually.

In general, as the number of individual tasks any given processor must handle is reduced, that processor's response time to new requests for service will be reduced. A well planned multiple processor bus structure will allow new processors to be added to the system in modular fashion. When new system functions (ie, more peripherals) are added, more processing power can be applied to handle them without impacting existing processor (master) task partitioning.

As used here, a "master" is any element existing on the system bus that may take control of the bus (ie, assert address and control lines). Typical examples include processors and direct memory access (DMA) controllers that address memory and input/output (I/O) locations resident on the bus. "Slave" elements include passive functions on the bus, such as memory or non-DMA I/O interfaces. Note that although slaves may possess intelli-

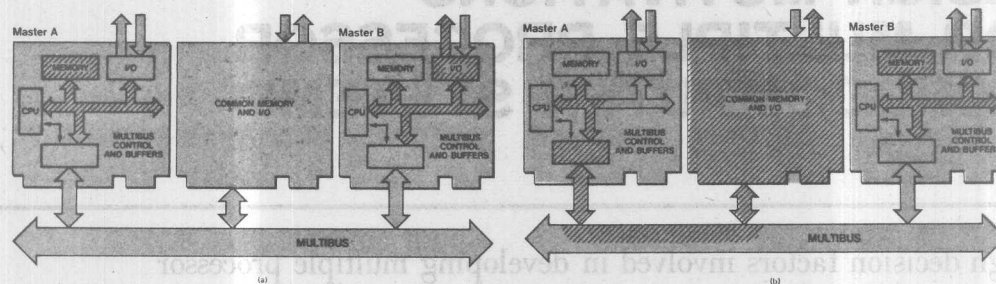


Fig 1 Multiple processor bus structure. Dual onboard/offboard structure of MULTIBUS allows each master to use its own memory and I/O without utilizing common system bus (a). Only when a master requires access to common memory or I/O does it use the bus (b). Note that other masters may continue onboard operations simultaneously

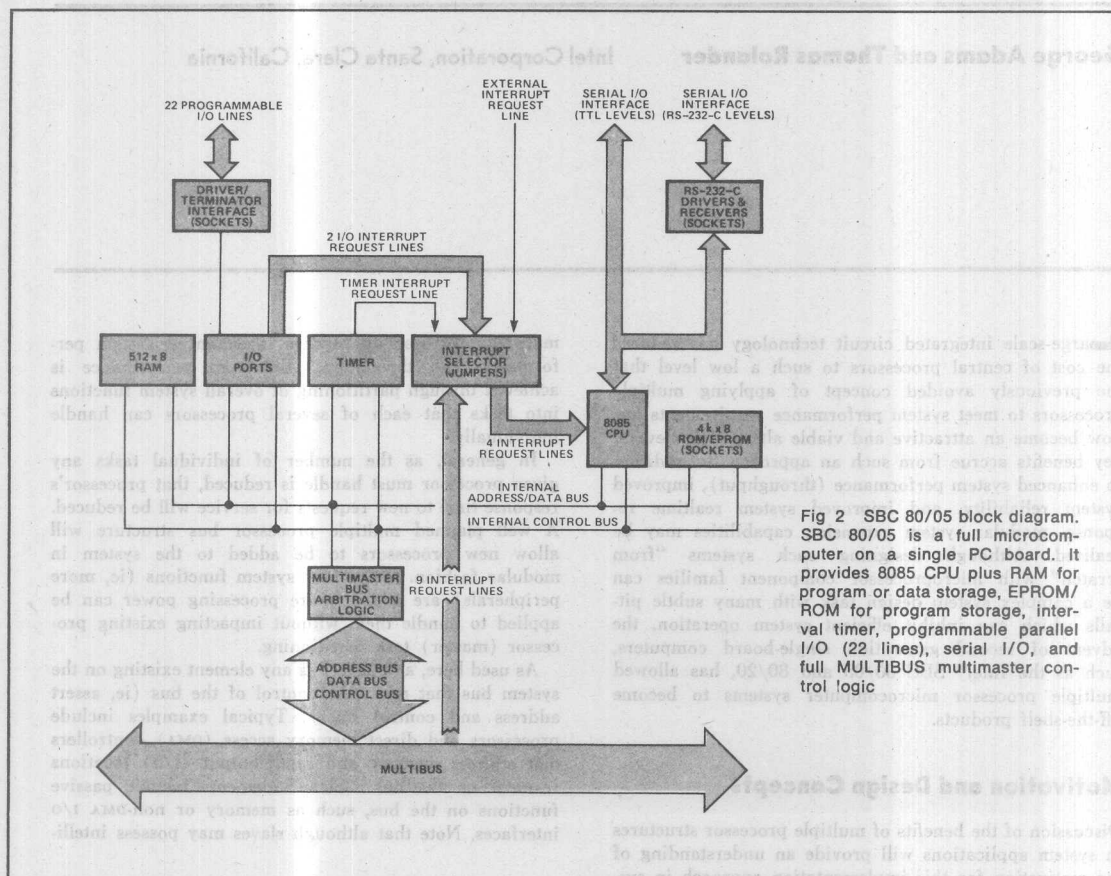


Fig 2 SBC 80/05 block diagram. SBC 80/05 is a full microcomputer on a single PC board. It provides 8085 CPU plus RAM for program or data storage, EPROM/ROM for program storage, interval timer, programmable parallel I/O (22 lines), serial I/O, and full MULTIBUS multimaster control logic

Hardware Considerations

Hardware considerations must be thoroughly evaluated in any multiple processor bus structure. These factors are described in detail around a specific implementation of such a structure, the Intel® MULTIBUS™, which supports multiple processor systems with its multi-master bus structure.

Bus Architecture

One architectural option open to the system designer is that of a multiple master/single bus structure. Under this partitioning, every master utilizes the common bus data path to fetch instructions or data from memory, read data from input devices, or write data to output devices or memory. Therefore, the common system bus rapidly becomes the bottleneck for overall system throughput, and fast DMA transfers can easily approach the full bandwidth of the bus during block transfers so that all other masters must idle for extended periods.

shown in Fig 1. Each processor-based master within the system retains its own local memory and I/O that it utilizes for most operations. Such local operations occur totally on the individual board and do not require the system bus. This greatly reduces the service request frequency by each master requiring use of the system bus. Such a dual-bus structure is implemented on the SBC 80/05 and 80/20 single-board computers, as shown in Figs 2 and 3, respectively, with the multi-master system bus (MULTIBUS).^{1,2}

Access to the system bus is requested only when a global (resident on the bus and accessible by multiple masters) memory location or I/O device is referenced during an instruction execution cycle. Local/global (on-board/offboard) distinction is defined through the value of the physical address referenced. If it lies within the address range of onboard memory or I/O, no bus request is made. Only when the address references a global

Intel® and MULTIBUS™ are trademarks of Intel Corp, Santa Clara, Calif.

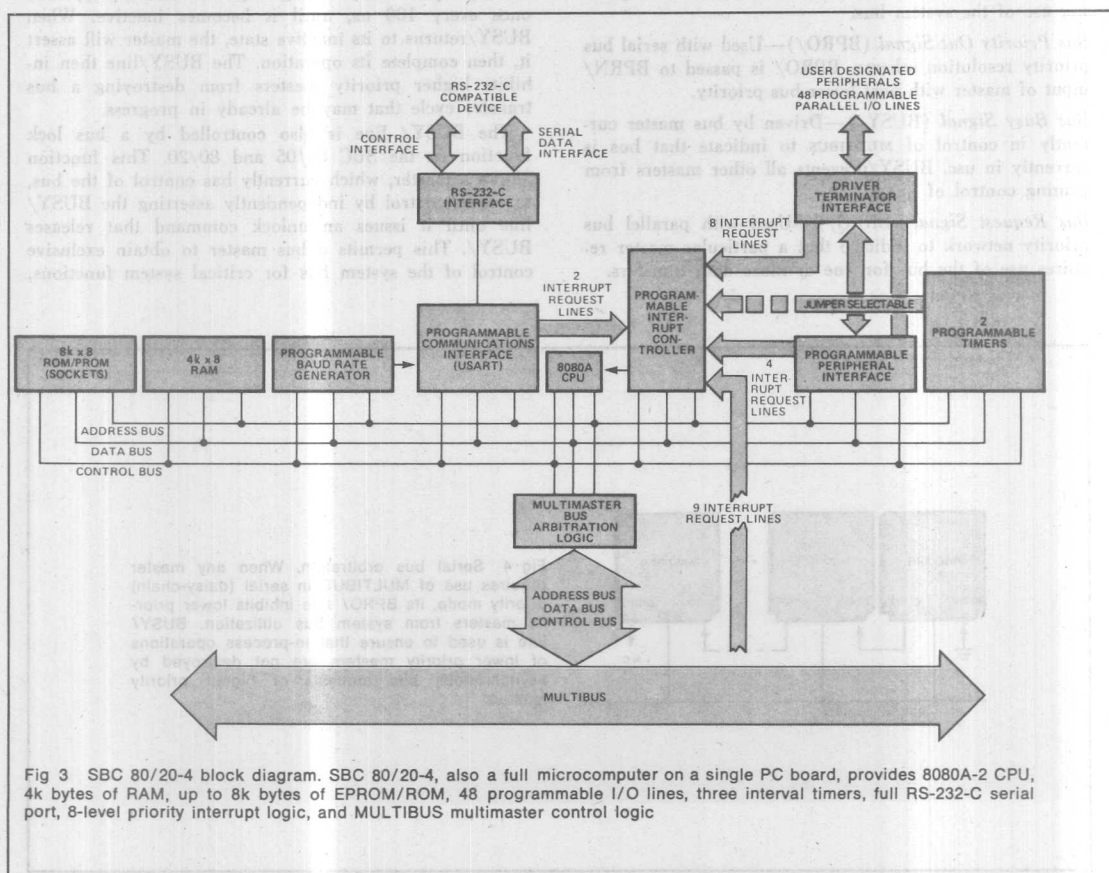


Fig 3 SBC 80/20-4 block diagram. SBC 80/20-4, also a full microcomputer on a single PC board, provides 8080A-2 CPU, 4k bytes of RAM, up to 8k bytes of EPROM/ROM, 48 programmable I/O lines, three interval timers, full RS-232-C serial port, 8-level priority interrupt logic, and MULTIBUS multimaster control logic

memory or I/O location, is a system bus request initiated. If no other master is currently utilizing the bus, this "new" master will be granted access immediately. However, this new master must wait if another master is currently utilizing the system bus. It continues to monitor the status of the system bus to determine when its current cycle may be completed. Thus, the MULTIBUS must provide a method for masters to determine whether or not another master is currently utilizing it.

Other masters may also simultaneously request the system bus. Arbitration must then be performed to resolve this multiple contention for the system bus. The MULTIBUS structure provides this arbitration in one of two techniques: serial (daisy chain) or parallel (encoded). The structure consists of four control lines that are synchronized by the common bus clock. These four control lines and the bus clock are active low. This is represented by the slash (/) character after each signal mnemonic. Control lines are as follows:

Bus Clock (BCLK/)—The negative edge of BCLK/ is used to synchronize bus arbitration. BCLK/ may be asynchronous to all CPU clocks, and it has a 100-ns minimum period. BCLK/ may be slowed, stopped, or single-stepped for debugging.

Bus Priority In Signal (BPRN/)—Indicates to a particular master that no higher priority master is requesting use of the system bus.

Bus Priority Out Signal (BPRO/)—Used with serial bus priority resolution scheme. BPRO/ is passed to BPRN/ input of master with next lower bus priority.

Bus Busy Signal (BUSY/)—Driven by bus master currently in control of MULTIBUS to indicate that bus is currently in use. BUSY/ prevents all other masters from gaining control of bus.

Bus Request Signal (BREQ/)—Used with parallel bus priority network to indicate that a particular master requires use of the bus for one or more data transfers.

Serial (Daisy-Chain) Bus Arbitration

In a serially arbitrated MULTIBUS system (Fig 4) requests for system bus utilization are ordered by priority on the basis of bus location. Each master on the bus notifies the next lower priority master when it needs to use the bus for a data transfer, and it monitors the bus request status of the next higher priority master. Thus the masters pass bus requests along from one to the next in a daisy-chain fashion.

The highest priority master (Master 1) in the system will always receive access to the system bus when it requires it. There is no higher priority master to inhibit its bus requests, and its bus priority input line (BPRN/) is thus permanently enabled.

Masters operate asynchronously on the MULTIBUS. A master may thus be in the middle of a bus operation when a higher priority master requests the bus. Obviously, interruption of such an in-process cycle must not be allowed. The mechanism for avoiding such erroneous operation is the BUSY/ line. Upon being notified that access to the bus is possible, the master examines BUSY/. If this control line is inactive, the master will assert it, and complete its bus operation. If BUSY/ is already active, another master is currently using the bus. In this case, the master will examine BUSY/ upon every falling edge of BCLK/, typically once every 100 ns, until it becomes inactive. When BUSY/ returns to its inactive state, the master will assert it, then complete its operation. The BUSY/ line then inhibits higher priority masters from destroying a bus transfer cycle that may be already in progress.

The BUSY/ line is also controlled by a bus lock function on the SBC 80/05 and 80/20. This function allows a master, which currently has control of the bus, to retain control by independently asserting the BUSY/ line until it issues an unlock command that releases BUSY/. This permits a bus master to obtain exclusive control of the system bus for critical system functions,

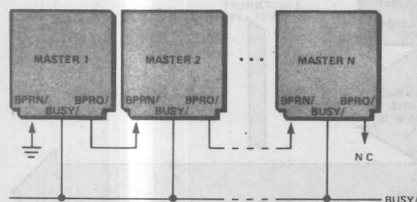


Fig 4 Serial bus arbitration. When any master requires use of MULTIBUS in serial (daisy-chain) priority mode, its BPRO/ line inhibits lower priority masters from system bus utilization. BUSY/ line is used to ensure that in-process operations of lower priority masters are not destroyed by asynchronous bus requests of higher priority masters

such as high speed memory or I/O data transfers and critical read-modify-write operations. With BUSY/ asserted in this way, all other masters will find the bus "in use" when they attempt to access it. Whereas system bus transfers normally take place on an interleaved basis (bus arbitration performed for each cycle), this bus lock function permits fast multiple-word transfers, when needed.

Two basic parameters determine the number of masters that can coexist on the system bus in serial bus arbitration mode. These are the BCLK/ cycle time and the BPRN/ to BPRO/ propagation delay of bus masters. Masters may be added to a system as long as the cumulative BPRN/ to BPRO/ propagation delay is such that the lowest priority master will always have its BPRN/ line driven inactive before the next BCLK/ falling edge after the highest priority master requests the bus. This worst-case timing condition is met as long as the following relationship is satisfied.

$$\sum_{i=1}^{N-1} (t_{BPRN-BPRO})_i < t_{BCLK} - t_{sh}$$

where

$(t_{BPRN-BPRO})_i$ = Propagation delay for master i
 t_{BCLK} = Bus clock (BCLK) cycle time (period)
 t_{sh} = Allowance for bus setup and hold times
 N = Number of bus masters

Using serial bus arbitration and SBC 80 onboard clocks, up to three masters may coexist on the system bus. This number can easily be extended, if desired, by generating a BCLK with a longer cycle. The SBC 80/05 and 80/20 provide a jumper option which allows the onboard BCLK/ to be disabled. This allows the system designer to generate BCLK/ externally.

Parallel (Hardware-Encoded) Bus Arbitration

The parallel bus arbitration technique resolves system bus master priorities using external hardware. The

parallel multimaster control line (BREQ/) comes into force in this case. Each master asserts BREQ/ when it requires access to the system bus. These lines are fed to a 2-chip parallel priority network. As with serial priority resolution, BPRN/ acts as the bus access enable input to each master. As Fig 5 illustrates, up to eight master priority levels are encoded by a 74148 priority encoder to a 3-bit code representing the highest priority master currently requesting the system bus. This code drives the 8205 3-to-8 decoder which asserts the proper BPRN/ line low to grant bus access to the highest priority master. The 74148/8205 propagation delay is less than 40 ns, easily fast enough to allow eight masters to coexist in this configuration utilizing a BCLK/ with a 100-ns period.

Systems requiring up to 16 masters may implement bus arbitration by utilizing two 74148 priority encoders and two 8205 decoders to provide a 16-level hardware priority network. The actual number of bus masters feasible on the system bus will also depend on bus drive/loading considerations. Even under this consideration, systems containing up to 16 masters are feasible.

Thus, single-board computer masters, in conjunction with the MULTIBUS control structure, provide off-the-shelf hardware solutions for the development of efficient multiple processor microcomputer systems. In addition to this hardware capability, the system designer needs to consider several software design issues.

Software Considerations

Several software operations, such as mutual exclusion, communication, and synchronization, are essential to proper multiple processor system operation. The MULTIBUS/SBC 80 functions that enable these software operations are examined.

Mutual Exclusion

In a multiple processor microcomputer system, there are

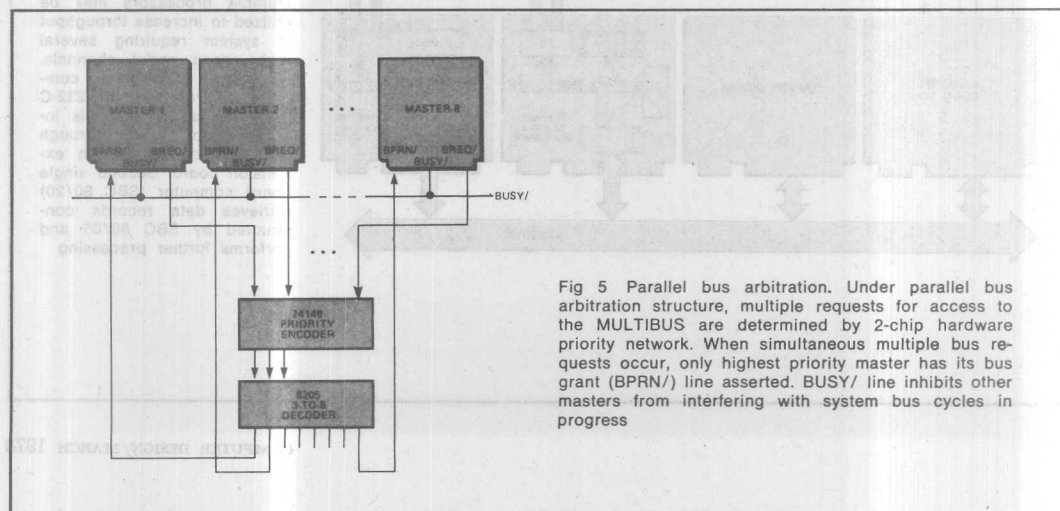


Fig 5 Parallel bus arbitration. Under parallel bus arbitration structure, multiple requests for access to the MULTIBUS are determined by 2-chip hardware priority network. When simultaneous multiple bus requests occur, only highest priority master has its bus grant (BPRN/) line asserted. BUSY/ line inhibits other masters from interfering with system bus cycles in progress

a mechanism to guarantee that asynchronous access to those resources is controlled in order to protect data from simultaneous change by two or more processors. Thus, some form of mutual exclusion must be provided to enable one processor to lock out access of a shared resource by other processors when it is in a critical section. A critical section is a code segment that once begun must complete execution before it, or another critical section that accesses the same shared resource, can be executed.

A Boolean variable can be used to indicate whether a processor is currently in a particular critical section (true) or not (false). Testing and setting this variable also presents a critical section. This function must be performed as a single indivisible operation; if it is not, two or more processors may test the variable simultaneously and then each set it, allowing them to enter the critical section at the same time. Such simultaneous entry would destroy the integrity of data and control parameters in global memory or cause erroneous double initialization of a global peripheral controller.

Mutual exclusion can be implemented as a software function alone, as described by Dijkstra⁴, for n processors operating in parallel. The SBC 80/05 and 80/20 bus lock function mentioned earlier provides a means for using program control to simplify mutual exclusion. While the system bus is locked, the master can perform the indivisible test and set operation on the Boolean

Communication

Communication is an essential function that allows a program executing on one processor to send or receive data from a program executing on another processor. Typically, two processors communicate through buffer storage in common memory. One program, called a producer, adds data to buffer storage; another, called a consumer, removes information from buffer storage.

In a typical application, one master may produce buffers of data that are to be consumed by a program executing on another master that services an output device. Communication through buffer storage requires the operations of adding to and taking from buffers. These operations constitute critical sections that can be controlled by providing mutual exclusion around the buffer manipulation operations.

Synchronization

At times there is a need for one master to send a synchronization signal to another. In a sense, synchronization is a special case of communication during which no data is transferred. Rather, the act of signaling is used to "wake up" a program executing on another master. A program may "sleep," by waiting for a synchronizing signal, until it receives a wake-up signal that enables it to continue execution. Manipulation of synchronization signals requires mutual exclusion.

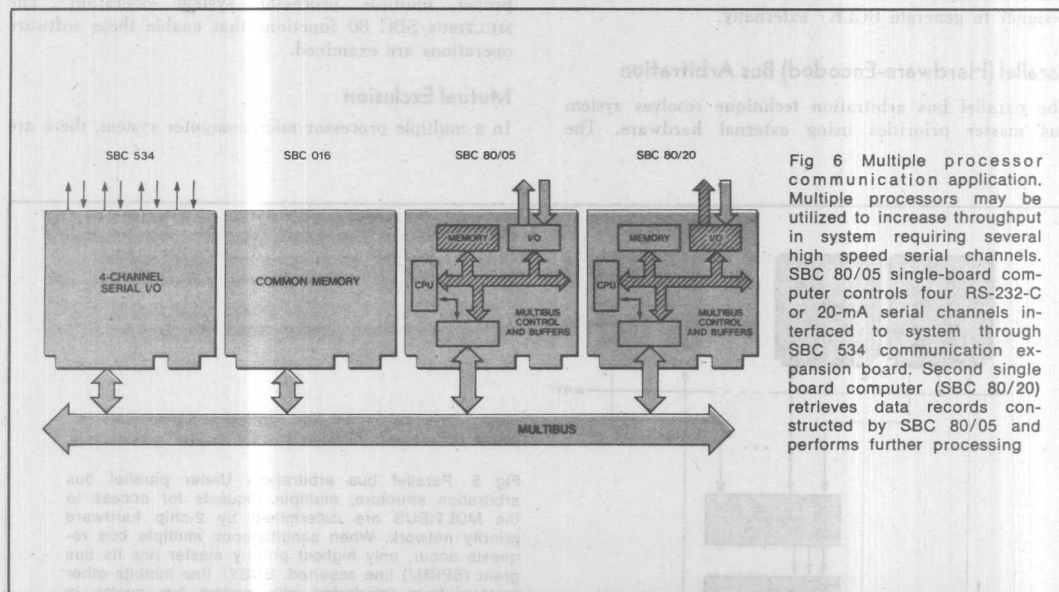


Fig 6 Multiple processor communication application. Multiple processors may be utilized to increase throughput in system requiring several high speed serial channels. SBC 80/05 single-board computer controls four RS-232-C or 20-mA serial channels interfaced to system through SBC 534 communication expansion board. Second single board computer (SBC 80/20) retrieves data records constructed by SBC 80/05 and performs further processing

System Initialization

In a microcomputer system that has multiple processors sharing a common system bus, a system initialization mechanism must be designed to set up the variables that control access to the shared resources. All single-board computers on the MULTIBUS begin execution simultaneously following a system reset. The bus lock function of the computers can be used by one specifically designated master to lock the bus immediately upon system reset and to perform system initialization for common resources before any other master attempts to access them. Since a locked bus has no effect on a single-board computer that is executing out of its local memory and using its local I/O, normal initialization by each processor can proceed while the shared resource initialization takes place.

Multiprocessor Applications

Two applications that are well suited to multiple processor microcomputer systems are examined. The first provides increased throughput, and the second allows shared resources.

Increased Throughput

Consider a system that is controlling multiple high speed

serial communication channels in addition to other data processing activities. In this case, multiple processors may be utilized to increase system throughput. Such a system with four full-duplex serial channels operating at 4800 baud could produce interrupts every 250 μ s. Interrupts at that frequency in a single master system would leave little time for other processing activities. In a multiple processor approach, one processor can be used to handle the interrupts from the serial channels, accumulate data into records, and then provide those records to another processor by placing them in common memory. The second processor is not burdened with the overhead of handling each character on an interrupt-driven basis, instead it is sent entire records of data available for further processing.

As shown in Fig 6, this application can be handled on the MULTIBUS with four boards. The SBC 80/05 single-board computer is used to service the communication board and prepare the data records. A 4-channel serial communication board (SBC 534) is used to provide the hardware interface for four serial communication channels. The SBC 80/20 single-board computer is used to process data records prepared by the SBC 80/05. Common memory is provided by the SBC 016 16k random-access memory (RAM).

Application of multiple processors to this problem requires communication through buffer storage. Two primitive operations, introduced by Dijkstra⁴, can be used to simplify the communication and synchronization between the masters. These primitives, designated P and V, operate on non-negative integer variables called

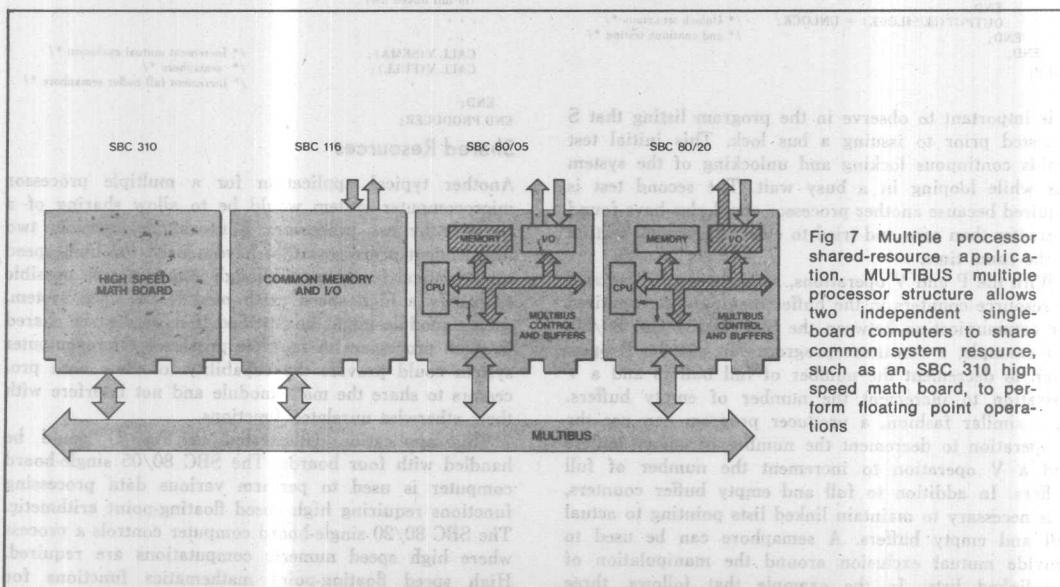


Fig 7 Multiple processor shared-resource application. MULTIBUS multiple processor structure allows two independent single-board computers to share common system resource, such as an SBC 310 high speed math board, to perform floating point operations

semaphores. The V procedure increments the semaphore (S) in a single indivisible operation. To make certain that fetch, increment, and store are not interrupted by another processor, the bus is locked during the operation.

Procedures for P and V primitive operations can be implemented in PL/M⁶ as follows:

```
V:
  PROCEDURE (S$ADR);
  DECLARE S BASED S$ADR BYTE;
  OUTPUT(BUS$LOCK) = LOCK; /* Lock MULTIBUS */
  S = S+1; /* Increment semaphore */
  OUTPUT(BUS$LOCK) = UNLOCK; /* Unlock MULTIBUS */
END V;
```

The P procedure loops in a busy wait until S is greater than zero, at which time it decrements S. The act of fetching, testing, decrementing, and storing S is also an indivisible operation. Note that if several masters with different speeds are in a busy wait on the same semaphore, the solution presented may not be "fair" to the lower speed processor; that is, the lower speed processor would test the semaphore less frequently, resulting in an unfair advantage for higher speed processors.

Implementation of a procedure for the P primitive is shown in the following PL/M code.

```
P:
  PROCEDURE (S$ADR);
  DECLARE S BASED S$ADR BYTE;
  DO FOREVER;
  IF S > 0 THEN /* Test semaphore */
  DO;
  OUTPUT(BUS$LOCK) = LOCK; /* Lock MULTIBUS */
  IF S > 0 THEN /* Retest semaphore */
  -DO;
  S = S-1; /* Decrement semaphore */
  OUTPUT(BUS$LOCK) = UNLOCK; /* Unlock MULTIBUS */
  RETURN; /* Exit from P procedure */
  END;
  OUTPUT(BUS$LOCK) = UNLOCK; /* Unlock MULTIBUS
  and continue testing */
  END;
END P;
```

It is important to observe in the program listing that S is tested prior to issuing a bus lock. This initial test avoids continuous locking and unlocking of the system bus while looping in a busy wait. The second test is required because another processor could also have found S greater than zero and tried to enter the critical section at the same time.

With the P and V operations, semaphores can be used as resource counters in the buffer manipulation required for communication between the SBC 80/05 and 80/20. For example, a consumer program can use the P operation to decrement the number of full buffers and a V operation to increment the number of empty buffers. In a similar fashion, a producer program can use the P operation to decrement the number of empty buffers and a V operation to increment the number of full buffers. In addition to full and empty buffer counters, it is necessary to maintain linked lists pointing to actual full and empty buffers. A semaphore can be used to provide mutual exclusion around the manipulation of the linked lists. In the example that follows, three variables (FULL, EMPTY, and SEMA) are used to implement these functions. The two PL/M programs illustrate consumer and producer code segments, respectively. Note that the consumer performs initialization because it accesses the semaphores prior to the producer.

```
CONSUMER:
  DECLARE EMPTY BYTE EXTERNAL; /* Number of empty buffers */
  DECLARE FULL BYTE EXTERNAL; /* Number of full buffers */
  DECLARE SEMA BYTE EXTERNAL; /* Binary semaphore */
  OUTPUT(BUS$LOCK) = LOCK; /* Lock MULTIBUS */
  EMPTY = NUMB$BUFFERS; /* Initialize semaphores */
  FULL = 0;
  SEMA = 1;
  OUTPUT(BUS$LOCK) = UNLOCK; /* Unlock MULTIBUS */
  DO FOREVER;
  CALL P(FULL); /* Decrement full buffer */
  CALL P(SEMA); /* semaphore */
  /* Decrement mutual exclusion */
  /* semaphore */
  (Take data from buffer and
  place it in local memory,
  move buffer from full to
  empty linked list)
  CALL V(SEMA); /* Increment mutual exclusion */
  CALL V(EMPTY); /* semaphore */
  /* Increment empty buffer */
  /* semaphore */
  (Process the data)
  END;
END CONSUMER;

PRODUCER:
  DECLARE (EMPTY, FULL, SEMA) BYTE EXTERNAL;
  DO FOREVER;
  (Prepare data in local
  memory)
  CALL P(EMPTY); /* Decrement empty buffer semaphore */
  CALL P(SEMA); /* Decrement mutual exclusion */
  /* semaphore */
  (Place data in a buffer,
  move buffer from empty
  to full linked list)
  CALL V(SEMA); /* Increment mutual exclusion */
  CALL V(FULL); /* semaphore */
  /* Increment full buffer semaphore */
  END;
END PRODUCER;
```

Shared Resources

Another typical application for a multiple processor microcomputer system would be to allow sharing of a resource by two processors. For example, consider two independent processors that have a need for high speed mathematical functions. Although it may not be possible to justify a high speed math module for each system, such a module might be justified if it were to be shared by both processors. A multiple processor microcomputer system could provide the capability to allow both processors to share the math module and not interfere with their otherwise unrelated functions.

This application (illustrated in Fig 7) could be handled with four boards. The SBC 80/05 single-board computer is used to perform various data processing functions requiring high speed floating-point arithmetic. The SBC 80/20 single-board computer controls a process where high speed numeric computations are required. High speed floating-point mathematics functions for both single-board computers are performed by an SBC 310 high speed math unit. SBC 116 combination memory and I/O board provides 16k RAM, 8k electrically programmable read-only memory (EPROM), 48 parallel I/O lines, and an RS-232-C serial port.

The problem to be solved in this application is to ensure that only one processor has access to the shared math module resource at one time. Thus, mutual exclusion must be provided to control the access to the resource. The following PL/M function returns TRUE if access to a critical section, used to implement the mutual exclusion, has been granted.

```

ENTER$CRITICAL$SECTION:
  PROCEDURE (FLAG$ADR) BYTE;
  DECLARE FLAG BASED FLAG$ADR BYTE;
  DECLARE ACCESS BYTE;
  IF FLAG = BUSY THEN
    RETURN FALSE;
  ACCESS = FALSE;
  OUTPUT(BUS$LOCK) = LOCK;
  IF FLAG = NOT BUSY THEN
    DO:
      FLAG = BUSY;
      ACCESS = TRUE;
    END;
    OUTPUT(BUS$LOCK) = UNLOCK;
    RETURN ACCESS;
  /* Test flag */
  /* Return false if busy */
  /* Lock MULTIBUS */
  /* Retest flag */
  /* Set flag busy */
  /* and access TRUE */
  /* Unlock MULTIBUS */
  /* Return either TRUE or */
  /* FALSE access */

```

END ENTER\$CRITICAL\$SECTION;

This PL/M function first tests the flag for the busy condition before issuing a busy lock. As in the P procedure described earlier, this initial test avoids continuous locking and unlocking of the MULTIBUS while a busy wait is being executed. The following procedure performs a busy wait operation on the flag used to control access to a critical section.

```

BUSY$WAIT:
  PROCEDURE (FLAG$ADR);
  DO WHILE NOT ENTER$CRITICAL$SECTION(FLAG$ADR);
  END;
END BUSY$WAIT;

```

Typical code segments illustrating the use of these procedures follow.

```

DECLARE MATH$BD$FLAG BOOLEAN EXTERNAL; /* Flag must be */
/* initialized */
MATH$BD$FLAG = NOT BUSY;
...
...
CALL BUSY$WAIT(MATH$BD$FLAG); /* Here we wait until */
/* we have access */
...
(Process math functions)
...
MATH$BD$FLAG = NOT BUSY; /* Set flag not busy */
...
/* We could also test and then do some other */
/* processing if the math module is busy */
IF ENTER$CRITICAL$SECTION(MATH$BD$FLAG)
THEN DO:
  ...
  ...
  (Process math functions)
  ...
  ...
  MATH$BD$FLAG = NOT BUSY; /* Set flag not busy */
END;
ELSE DO:
  ...
  ...
  (Something else)
  ...
  ...
END;

```

Conclusions

The motivations for implementing multiple processor microcomputer systems include enhanced performance

and throughput. When the appropriate hardware/software design considerations are made, modularity is easily achieved. Hardware solutions to many problems are provided by means of a MULTIBUS structure and SBC 80 single-board computers that have multimaster capability. Through control of MULTIBUS functions, the software designer can perform multiple processor communication, synchronization, and mutual exclusion.

Even with these significant steps toward the simplification of multiple processor microcomputer systems, the design of such systems remains a complex software/hardware design task. The future trend of multiple processor microcomputer systems will be to simplify the software tasks of implementing communications, synchronization, and mutual exclusion. These functions could be performed in varying degrees by additional hardware bus functions.

Potential rewards for a multiple processor architecture include enhanced system throughput, improved real-time response, modular system expansion, and improved system reliability. These benefits will pressure the technology of parallel processing to include microcomputers in an increasing number of computer applications.

References

1. "SBC 80/05 Hardware Reference Manual," Pub 9800483, Intel Corp, Santa Clara, Calif, 1977
2. "SBC 80/20 Hardware Reference Manual," Pub 9800317, Intel Corp, Santa Clara, Calif, 1976
3. A. C. Shaw, *The Logical Design of Operating Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1974, pp 59-78
4. E. W. Dijkstra, "Solution of a Problem in Concurrent Programming Control," *Communications of the ACM*, Sept 1965, p 569
5. "Intel MULTIBUS Interfacing," Pub AP-28, Intel Corp, Santa Clara, Calif, 1977
6. D. McCracken, *A Guide to PL/M Programming for Microcomputer Applications*, Addison-Wesley, Reading, Mass, 1978

George Adams, as product line manager for single-chip microcomputers with Intel, is responsible for marketing and applications engineering for MCS-48™ microcomputers. His experience includes work as a microcomputer applications specialist and product planner, and as a computer design engineer. He holds a BSEE from the University of Miami and an MBA from Boston University.

Thomas Rolander is currently a partner of Dharma Systems, a computer systems consulting firm, where he is involved in systems engineering, software, and hardware design. Previously he served as an applications engineering manager for OEM computer systems at Intel. He received a Bachelor's degree in civil engineering and a Master's degree in electrical engineering from the University of Washington.

September, 1978

Triple-bus architecture on a single-board microcomputer

By Jim Johnson, Craig Kinnie and Mike Maertz
Electronic Design 15 / July 19, 1978

- References**
1. "8080/8085 Hardware Reference Manual," Feb 1980.
 2. "8080/8085 Hardware Reference Manual," Feb 1980.
 3. "8080/8085 Hardware Reference Manual," Feb 1980.
 4. "8080/8085 Hardware Reference Manual," Feb 1980.
 5. "8080/8085 Hardware Reference Manual," Feb 1980.
 6. "8080/8085 Hardware Reference Manual," Feb 1980.
 7. "8080/8085 Hardware Reference Manual," Feb 1980.
 8. "8080/8085 Hardware Reference Manual," Feb 1980.
 9. "8080/8085 Hardware Reference Manual," Feb 1980.
 10. "8080/8085 Hardware Reference Manual," Feb 1980.

in an increasing number of computer applications.

hardware has functions.

could be performed in various devices by additional

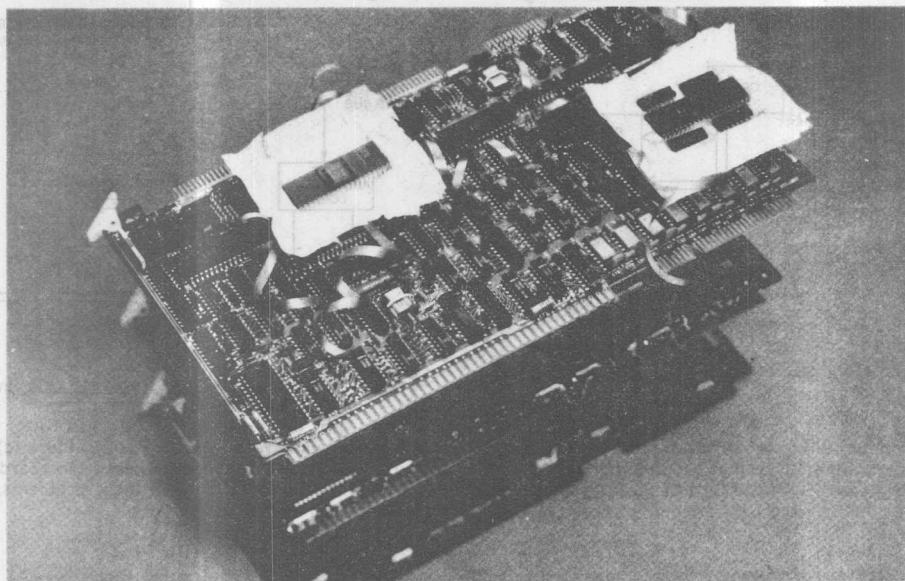
communication, synchronization, and mutual exclusion.

and throughout. When the appropriate hardware

control access to a critical section.

September, 1978

Triple-bus architecture lets a single-board microcomputer's CPU operate at full speed while other system components share the main memory.



The introduction of Intel's iSBC 80/30 marks the beginning of the third generation of single board computer architecture. Two features separate the new microcomputer from second-generation single-board μ Cs. The major one is a triple-bus architecture that supports a dual-port memory. As a result, the on-board CPU does not tie up the main system bus (Intel's Multibus) when using the memory. Moreover, with two ports, the memory becomes a global resource, accessible via the three buses from the on-board 8085A CPU as well as from remote CPUs and other external devices in multimaster schemes.

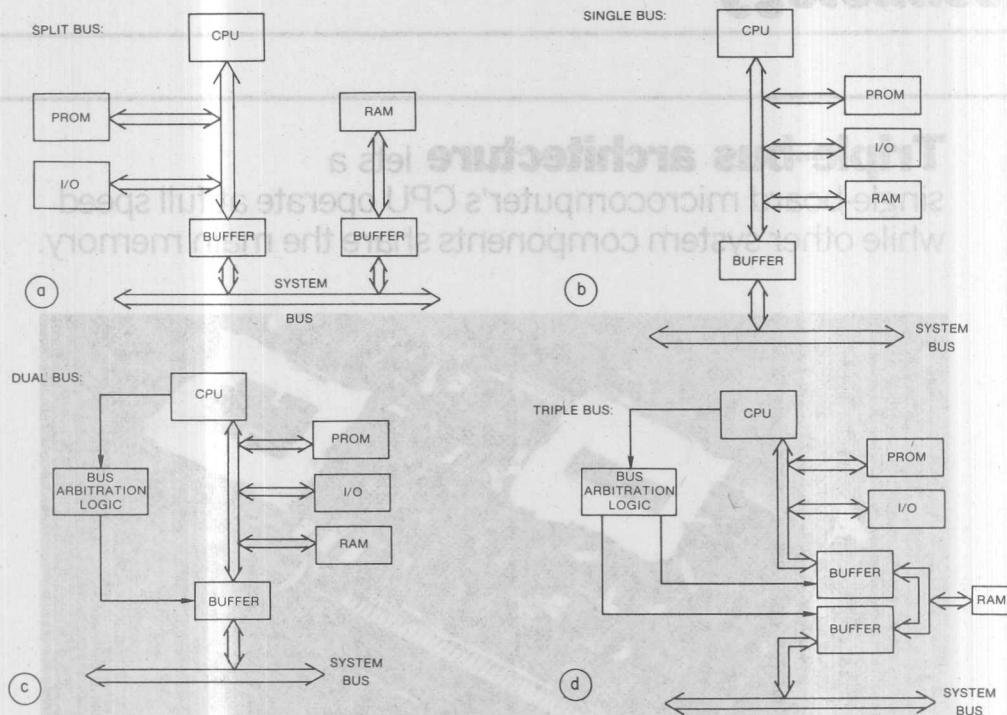
In addition, the 80/30 contains two microprocessors: an 8085A acting as the master CPU and an 8041 single-chip microprocessor acting as a slave, or intelligent I/O, processor.

Jim Johnson, Project Leader, **Craig Kinnie**, Project Manager, and **Mike Maerz**, Marketing Manager, Intel Corp., Santa Clara, CA 95051.

To appreciate the benefits of the 80/30's triple-bus, dual-port memory architecture, examine the following problem. Now that fully one fourth (16-kbytes) of the available memory space in a 64-kbyte μ C system can reside on a single-board μ C, the CPU must share these 16-kbytes with other system components, such as direct-memory-access devices, discs and other processors. What's the best solution—especially when, in many applications, 16-kbytes is all the memory that's required by the whole system?

Alternatives have problems

The most straightforward way is a split-bus architecture, in which both the CPU and the system have equal access to the memory (Fig. 1a). While the system bus will be able to handle memory access efficiently from devices tied to it, it will be tied up by the CPU—so external operations not related to memory accesses will be hindered.



1. **Microcomputer-bus organizations** takes several forms: In a split-bus approach (a) the CPU and system have equal access to memory, but the CPU ties up the system bus; in a single-bus (b), the CPU encounters extra delays in

using the system bus. A dual-bus structure (c) also has buffer delays, and no system access to on-board memory. But a triple-bus (d) avoids all these problems, allowing total system access to memory.

A single-bus approach (Fig. 1b) is hampered by buffer and bus-intervention delays which limit the CPU's performance. And dual-bus architecture (Fig. 1c), while granting the CPU exclusive access, does not allow other bus masters access to the memory. Also dual-bus suffers from buffer delays.

A triple-bus, dual-port architecture (Fig. 1d) provides the benefit of both single and dual-bus architectures: total system access and exclusive access by the CPU. But it also has its disadvantages: Dual-port architecture requires many buffers as well as access-arbitration logic. However, 20-pin octal buffers introduced by several manufacturers don't take up nearly as much board space or cost as much as equivalent standard buffers. Since the octal buffers come in unidirectional or bidirectional forms—and at nearly the same cost—the three-bus approach used on the 80/30 actually takes only as many packages as the split-bus approach.

Access arbitration is solved in the 80/30 with cycle status signals from the 8085A CPU. Instead of providing equal access to the RAM from both the CPU and the system, the arbitration logic is designed to favor the CPU. By assigning the default state of the arbiter

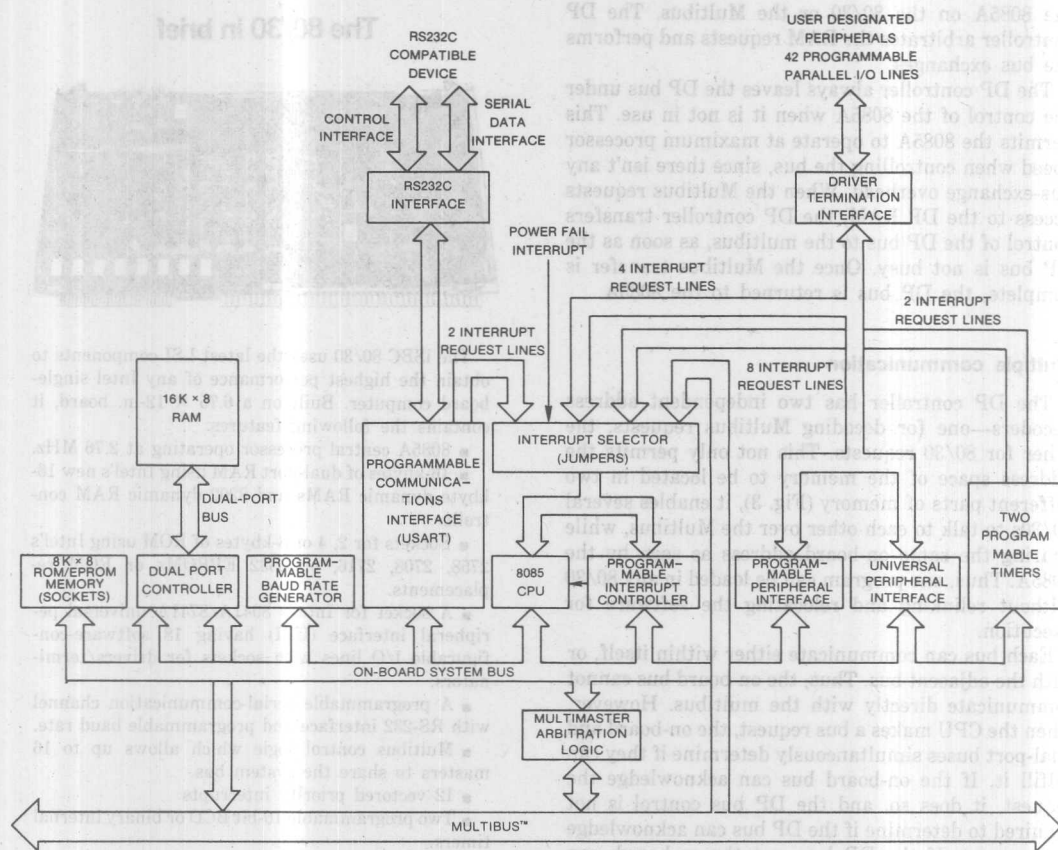
to the CPU, the logic anticipates a CPU memory access and reserves the memory until the cycle is complete.

In addition, if an on-board CPU access is imminent, a reservation signal derived from the 8085A CPU status signals, the ALE (address latch enable), the address, and the cycle status signals (SO, SI, IO/M) will hold off bus contention. As a result, the CPU can operate at full speed without tying up the system bus.

Of course, this extra CPU performance cuts into the rest of the system's memory-access time. However, the penalty imposed by the arbiter is less than 200 ns—less than the time it would take a DMA device to regain control of the bus in the split-bus approach, where access must be interleaved.

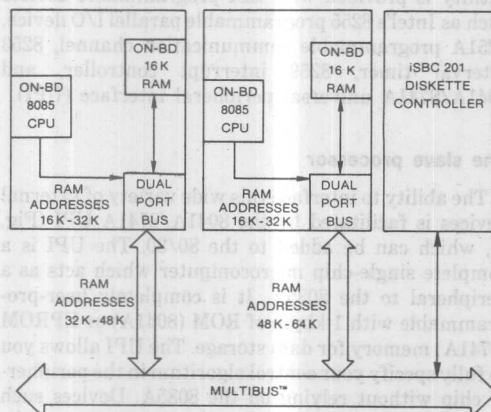
A bus hierarchy

The three buses in the 80/30 hierarchy (Fig. 2) are an on-board bus, a dual-port (DP) bus and the Multi-bus (system bus). Innermost is the on-board bus, which connects the 8085A, all on-board I/O peripherals and ROM. The next bus in the hierarchy, the dual-port connects a dual-port controller, 16-kbytes of dynamic RAM and a dynamic RAM controller. The



2. The full 80/30 one-board microcomputer is organized around its three buses: on board, dual-port, and the external-system Multibus. The main CPU, an 8085A, runs

at 2.76 MHz, while an 8041A one-chip microprocessor serves as a peripheral controller or slave processor, running with a 2.6-ms cycle time.



3. The microcomputer's on-board memory may be addressed independently by the on-board central processor and Multibus bus masters to increase the efficiency of usage of the total available memory space.

outermost bus, the Multibus, offers modules that permit either the expansion or addition of system resources.

With the on-board bus, the 8085A communicates with its on-board I/O and ROM (or PROM, if desirable) and the dual-port bus. Since the on-board bus permits access to the I/O and ROM only from the 8085A, all I/O and ROM (up to 8-kbytes are the 8085A's private property). And as a result, the 80/30 can operate on its on-board bus while another Multibus master uses the Multibus, accessing data from the board's dual-port RAM without reducing processor speed.

The dual-port (DP) bus contains 16-k of read/write memory, implemented with Intel's 2117 16-kbyte dynamic RAM and the 8202 dynamic RAM controller (DRC). The DRC interfaces the DP bus to the 16-kbytes of dynamic RAM, and provides an almost static-RAM type interface. It provides the system with multiplexed addresses, address strobes, and refresh control to the RAM, as well as refresh/access arbitration and acknowledgments.

The RAM on this bus can be accessed from either

the 8085A on the 80/30 or the Multibus. The DP controller arbitrates the RAM requests and performs the bus exchanges.

The DP controller always leaves the DP bus under the control of the 8085A when it is not in use. This permits the 8085A to operate at maximum processor speed when controlling the bus, since there isn't any bus-exchange overhead. When the Multibus requests access to the DP RAM, the DP controller transfers control of the DP bus to the multibus, as soon as the DP bus is not busy. Once the Multibus transfer is complete, the DP bus is returned to the 8085A.

Multiple communication

The DP controller has two independent address decoders—one for decoding Multibus requests, the other for 80/30 requests. This not only permits the address space of the memory to be located in two different parts of memory (Fig. 3), it enables several 80/30s to talk to each other over the Multibus, while sharing the same on-board address as seen by the 8085A. Thus, one program can be loaded in any 80/30 without relinking and relocating the software for execution.

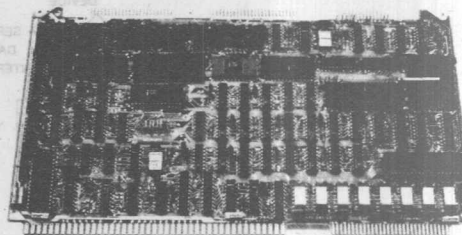
Each bus can communicate either within itself, or with the adjacent bus. Thus, the on-board bus cannot communicate directly with the multibus. However, when the CPU makes a bus request, the on-board and dual-port buses simultaneously determine if they can fulfill it. If the on-board bus can acknowledge the request, it does so, and the DP bus control is not required to determine if the DP bus can acknowledge the request. If the DP bus, not the on-board, can acknowledge the request, it does so, and the controller then lets the CPU use the bus. Thereafter, the RAM controller completes the operation and generates an acknowledge signal.

If neither the on-board nor DP bus can fill the bill, the Multibus is solicited by the CPU. Since a bus can only communicate with an adjacent bus, the on-board bus must request the DP bus to communicate with the Multibus via the DP controller. The on-board bus will retake control of the DP bus only after the request to use the Multibus is granted. This prevents lockout problems with the DP bus, where the CPU requests the Multibus when it is controlled by another bus master accessing the DP RAM.

How the 80/30 performs is directly related to how many buses it must use to complete a requested operation. The on-board bus always operates at maximum processor speed. The DP bus operates at maximum only if it hasn't been busy and a memory refresh cycle was not in process. The processor speed when the Multibus is used depends on bus overhead involved and the type of module requested.

The 80/30 boasts more than a three-bus architecture. For one thing, its I/O is designed to interface to a wide variety of external devices, including switches, motor drives, bistable sensors, displays,

The 80/30 in brief



The iSBC 80/30 uses the latest LSI components to obtain the highest performance of any Intel single-board computer. Built on a 6.75 × 12-in. board, it contains the following features:

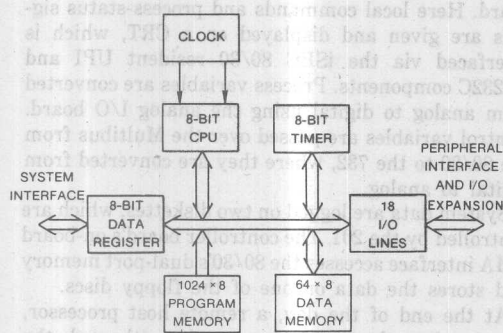
- 8085A central processor operating at 2.76 MHz.
- 16-kbytes of dual-port RAM using Intel's new 16-kbyte dynamic RAMs and 8202 dynamic RAM controller.
- Sockets for 2, 4 or 8-kbytes of ROM using Intel's 2758, 2708, 2716, or 2332 EPROMs or ROM replacements.
- A socket for Intel's 8041A/8741A universal peripheral interface (UPI) having 18 software-configurable I/O lines with sockets for drivers/terminators.
- A programmable serial-communication channel with RS-232 interface and programmable baud rate.
- Multibus control logic which allows up to 16 masters to share the system bus.
- 12 vectored priority interrupts.
- Two programmable 16-bit BCD or binary internal timers.

keyboards, printers, teletypewriters, communicator modems, cassettes and other computers. This versatility is provided with LSI programmable devices such as Intel's 8255 programmable parallel I/O device, 8251A programmable communication channel, 8253 interval timer, 8259 interrupt controller, and 8041A/8741A universal peripheral interface (UPI).

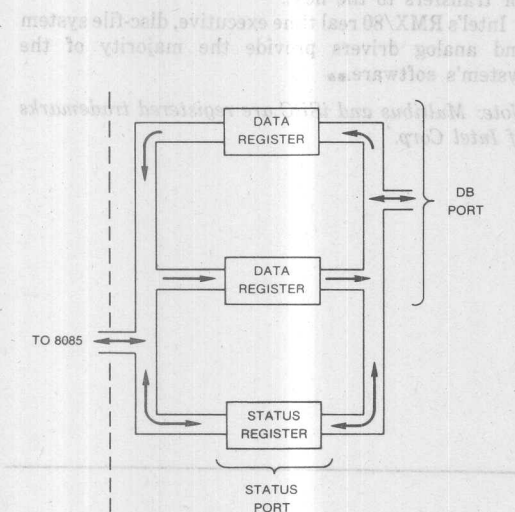
The slave processor

The ability to interface this wide variety of external devices is facilitated by the 8041A/8741A UPI (Fig. 4), which can be added to the 80/30. The UPI is a complete single-chip microcomputer which acts as a peripheral to the 8085A. It is completely user-programmable with 1-kbyte of ROM (8041A) or EPROM (8741A) memory for data storage. The UPI allows you to fully specify your control algorithm in the peripheral chip without relying on the 8085A. Devices such as printer controllers and keyboard scanners can be completely self-contained, relying on the 8085A only for data transfers.

The UPI is a powerful 8-bit CPU with a 2.6-ms cycle time and an instruction set optimized for bit manipu-



4. The 8041A/8741A single-chip microcomputer (UPI-41) has its own on-chip ROM and RAM and can be programmed to perform various peripheral control functions.



5. The UPI's two data registers are organized so that the 8085A CPU can write in just one register and read from the other. As a result, the two registers appear as one register to the main 8085A CPU.

lation and I/O operations. It contains an 8-bit counter/timer, buffers to communicate with the 8085A, and two 8-bit programmable I/O ports, which can be customized by software or by plugging in suitable line drivers or terminators into sockets. The UPI also has two input bits that it can test directly. An RS-232 driver and receiver on the 80/30 permit the UPI to be programmed as a simple serial-communication channel.

Interfacing to the on-board bus

The UPI interfaces asynchronously with the on-board bus using two data and two status registers. The UPI's two internal data registers appear to the

8085A as only one register, since one data register can be written into only by the UPI and read only by the 8085A, and the other can be written into only by the 8085A and read by the UPI (Fig. 5). This is done to prevent the two CPUs from simultaneously writing into a data register.

The UPI can communicate with the 8085A by loading a data register and then returning to its previous control task. The 8085A can periodically poll the UPI status port for the valid-read (VR) flag, which is set in hardware when the UPI writes to its data port, or the UPI can generate an interrupt to the 8085A via an I/O bit that can be programmed to be the VR flag.

Once the 8085A determines the VR flag is true, it can transfer the data to its own memory without disturbing the UPI. The VR flag is automatically cleared after the data are transferred. Similarly, when the 8085A transfers data to the UPI, a valid output (VO) flag is set and an interrupt to the UPI is generated (if enabled) automatically. Once the UPI transfers the data, the VO flag is cleared. The VO flag can also be programmed to a port bit for generating interrupts to the 8085A to indicate that the transfer is complete.

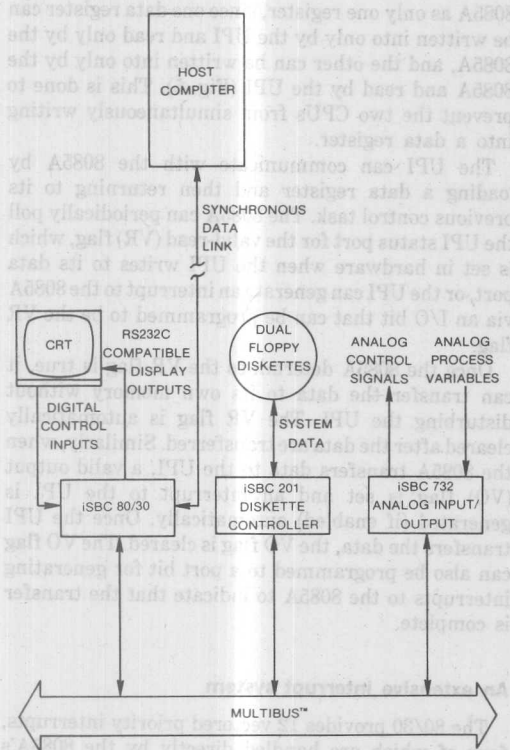
An extensive interrupt system

The 80/30 provides 12 vectored priority interrupts, four of which are handled directly by the 8085A's interrupt-processing capability and routed to fixed, unique memory locations. The remaining eight levels are handled via the 8259A programmable interrupt controller (PIC), which generates a unique memory address for each level. These addresses are equally spaced at intervals of four or eight (software-selectable) bytes. This 32 or 64-byte block may be located to begin at any 32 or 64-byte boundary in the 65,536-byte memory space. A single 8085A jump instruction at each of these addresses then provides the linkage to locate each interrupt-service routine independently anywhere in memory. The PIC provides a selection of four priority algorithms so that the manner in which real-time requests are processed may be configured to meet the requirements of the system under design.

The 80/30 also has two 8253-based programmable 16-bit BCD and binary timers/event counters, which can be used for a variety of functions. Both timers may be set to act as a rate generator (divide-by-N counter), a square-wave generator, a programmable retriggerable one-shot, or one of the timers can be jumper-selected as an event counter. In addition, an interrupt can be generated when a time interval has expired or when a specified number of events has occurred.

To see how useful the 80/30 can be, consider a supervisory control/monitoring system (Fig. 6) using an Intel iSBC 80/30 single-board computer, iSBC 201 diskette controller, and iSBC 732 analog input/output

ELECTRONIC DESIGN 15, July 19, 1978



6. In this application example, the 80/30 forms the heart of a remote data-acquisition system. By taking advantage of the one-board microcomputer's dual-port memory and universal peripheral interface, the system achieves a combination of attractive cost and efficiency.

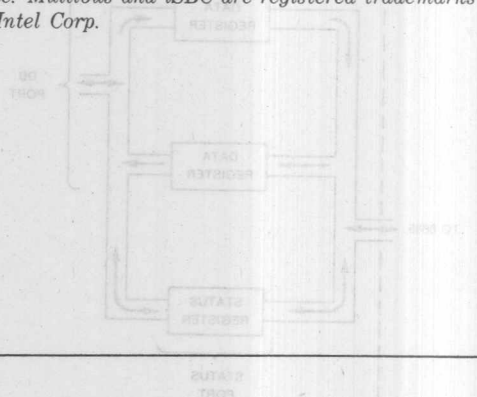
board. Here local commands and process-status signals are given and displayed on a CRT, which is interfaced via the iSBC 80/30 resident UPI and RS232C components. Process variables are converted from analog to digital using the analog I/O board. Control variables are passed over the Multibus from the 80/30 to the 732, where they are converted from digital to analog.

System data are logged on two diskettes, which are controlled by the 201. The controller board's on-board DMA interface accesses the 80/30's dual-port memory and stores the data on one of the floppy discs.

At the end of the day, a remote host processor, interfaced to the 80/30 via a modem (through the 80/30's 8251A and RC232C circuits) can request all or part of the diskette-resident data. Here, the 80/30 uses its on-board dual-port memory as a data buffer for transfers to the host.

Intel's RMX/80 real time executive, disc-file system and analog drivers provide the majority of the system's software.■

Note: Multibus and iSBC are registered trademarks of Intel Corp.



2. The UPI's two data registers are organized so that the 8085A CPU can write to one register and read from the other. As a result, the two registers appear as one register to the main 8085A CPU.

lation and I/O operations. It contains an 8-bit counter timer, buffers to communicate with the 8085A, and two 8-bit programmable I/O ports which can be configured by software or by pinning in suitable line drivers or terminators into modes. The UPI also has two input bits that it can test directly. An RS-232C driver and receiver on the 80/30 permit the UPI to be programmed as a simple serial communication channel.

Interfacing to the on-board bus

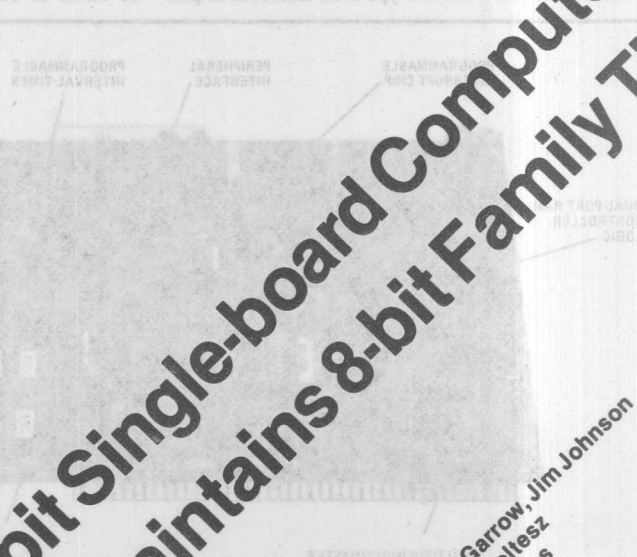
The UPI interfaces asynchronously with the on-board bus using two data and two status registers. The UPI's two internal data registers appear to the

November 1978

This enhanced processing power is supported by the largest memory ever fitted on a CPU board (Fig. 1). Memory address space has been extended over the 128K of 8086 to one million bytes (1 to 16 Kilobytes of

For the first time ever, 8- and 16-bit single-board computers can interoperate over the same system bus. The iSB-C 88412 16-bit SBC has been designed to work seamlessly with its predecessor, the iSB-C 80 family of 8-bit boards. What's in it for the user? Design flexibility—8-bit designs can be enhanced to 16-bit developments; can be transported and beyond that, 8- and 16-bit devices can be mixed in multiprocessing configurations. Several features make these options possible: a 16-bit CPU and instruction set designed for 8-bit compatibility; greatly expanded memory resources; and an extension of the Multibus specifications.

At the heart of the iSB-C 88412 is a 16-bit, high-performance, multi-odd-even-instruction-8086 central processing unit that operates at 2 megahertz. Because the 80-instruction set is a superset to that of both the 8085 and 8086, 8-bit processors, the CPU can execute the full 8085/8086 and 8-bit instructions plus



**16-bit Single-board Computer
Maintains 8-bit Family Ties**

By Robert Garrow, Jim Johnson
and Les Soltesz
Electronics

**By Robert Garrow, Jim Johnson
and Les Soltesz
Electronics**

Technical articles

16-bit single-board computer maintains 8-bit family ties

Three-bus 8086-based board addresses a megabyte, communicates over expanded system bus

by Robert Garrow, Jim Johnson, and Les Soltesz, Intel Corp., Santa Clara, Calif.

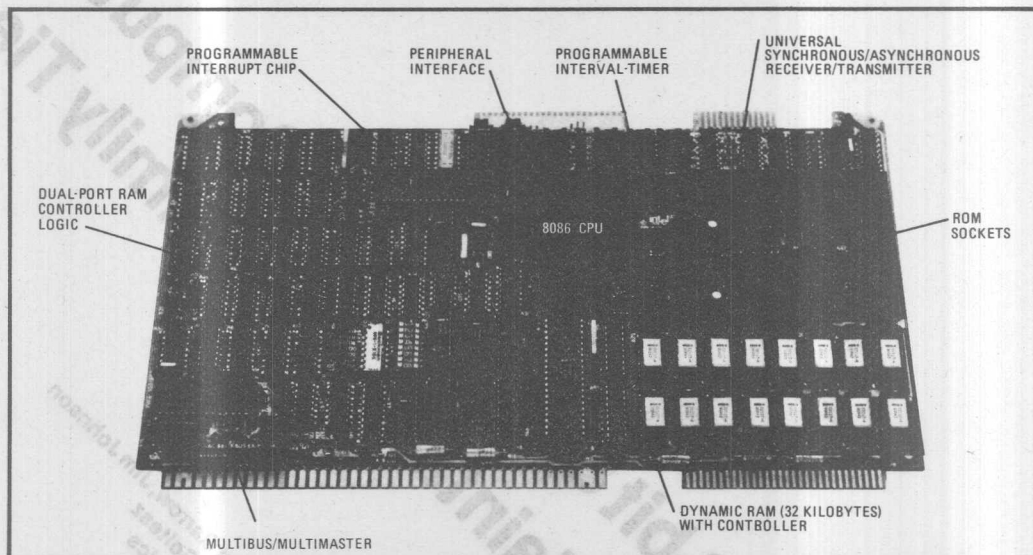
□ For the first time ever, 8- and 16-bit single-board computers can brainstorm over the same system bus. The iSBC 86/12 16-bit SBC has been designed to work intimately with its predecessors, the iSBC 80 family of 8-bit boards. What's in it for the user? Design flexibility—8-bit designs can be enhanced to 16 bits, developed software can be transported and, beyond that, 8- and 16-bit devices can be mixed in multiprocessing configurations. Several features make these options possible: a 16-bit CPU and instruction set designed for 8-bit compatibility; greatly expanded memory resources; and an extension of the Multibus specifications.

At the heart of the iSBC 86/12 is a 16-bit, high-performance metal-oxide-semiconductor 8086 central processing unit that operates at 5 megahertz. Because the 8086 instruction set is a superset to that of both the 8080A and 8085A 8-bit processors, the CPU can execute the full set of 8080A/8085A-type 8-bit instructions plus

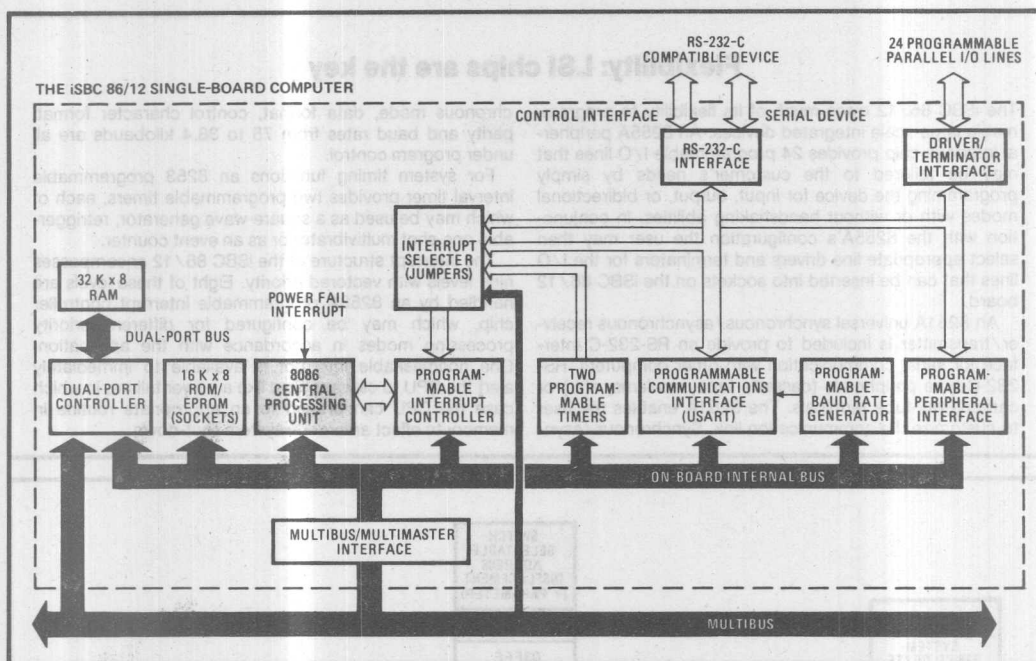
a new set of 16-bit instructions. Thus, programs generated for 8-bit-CPU systems can easily be upgraded to run on the iSBC 86/12 using the software tools available with the Intel microcomputer development system. Programs written in Intel's high-level programming language, PL/M, can be executed on both iSBC 80 and iSBC 86 products, preserving the software investment in 8-bit systems as a user moves into 16-bit applications.

Other features of the 8086 CPU are signed 8- and 16-bit arithmetic (including multiply and divide), efficient interruptible byte-string operations, and improved bit manipulation. Furthermore, the 8086 provides mechanisms for reentrant code, position-independent code, and dynamically relocatable programs.

This enhanced processing power is supported by the largest memory ever offered on a CPU board (Fig. 1). Memory address space has been extended over the iSBC 80 series to one million bytes. Up to 16 kilobytes of



1. What a board. The iSBC 86/12 has 32 kilobytes of RAM and room for 16 kilobytes of ROM. The 5-MHz 8086 CPU executes 8080A/8085A-type as well as 16-bit instructions, including multiply and divide. Address space has been increased to a megabyte.



2. LSI + SBC = 86/12. A number of programmable LSI devices take credit for the power and flexibility of the iSBC 86/12. Note their interconnection to the three-bus hierarchy. When the 8086 requests a resource, the system bus is used only as a last resort.

read-only memory can be installed on the iSBC 86/12 itself. Furthermore, an additional 32 kilobytes of dynamic random-access memory with on-board refresh may be accessed independently by the CPU or by the system bus (Multibus).

Like the iSBC 80/30, the 86/12's RAM has dual ports to extend its use off board for access by other Multibus masters, including single-board computers, direct-memory-access devices, and peripheral controllers [Electronics, Aug. 17, p. 109]. All memory operations on the board occur independently of the Multibus, freeing it for external parallel operations. For applications that require data integrity at all times, a separate bus supplies power to the RAM and support logic via the edge connector. An auxiliary power source energizes the RAM in the event of power failure.

Multibus—the new look

To exploit the greater performance of the 8086 CPU and simultaneously make the iSBC 86/12 fully compatible with the iSBC 80 family of SBCs and expansion products, the Multibus specification has been extended to support 20 bits of address and 16 bits of data. The control lines, too, have been expanded to direct 8- and 16-bit data transfer over the system bus. These improvements enable the iSBC 86/12 to address directly a full one megabyte of system memory, access data in 8- or 16-bit word lengths, and recognize and acknowledge a variety of interrupts.

Address space has been enlarged to 1 megabyte by adding four address lines, A₁₀–A₁₃. Next, 8- and 16-bit

data operations have been defined to permit both types in the same system. This is done by reorganizing the memory modules, adding one new signal and redefining another. The memory is divided into two 8-bit data banks, which form a single 16-bit word. The banks are organized such that all even-byte-addressed data is in one bank (D₀–D₇) and all odd-byte-addressed data is in the other bank (D₈–D_F). A new bus-address signal has been defined to control the odd-byte bank called byte high enable (BHEN) during 16-bit operations. When active, BHEN enables the high byte of the data word from the addressed boards on the D₈–D_F Multibus data lines. A₀ controls the even byte bank and, when inactive, enables the low byte of the data word on the D₀–D₇ Multibus data lines. All word operations must occur on an even-byte-address boundary with BHEN active for maximum efficiency. (A₀ is inactive for all even addresses—see the table.) Word operations on odd-byte boundaries will be converted to 2-byte operations by the 8086, one for low-byte, one for high-byte. Byte operations can occur in one of two ways. The even bank is accessed when BHEN and A₀ are both low. This puts the data on D₀–D₇. To access the odd bank (normally placed on D₈–D_F during a word operation), a new data path has been defined. The active state of A₀ and the inactive state of BHEN are used to enable a swap-byte buffer, which places the odd data bank on D₀–D₇. This permits an 8-bit master access to both bytes of the data word while controlling only A₀. A₀ therefore specifies a unique byte and is not part of the word address, since all word operations are on even-byte boundaries.

Flexibility: LSI chips are the key

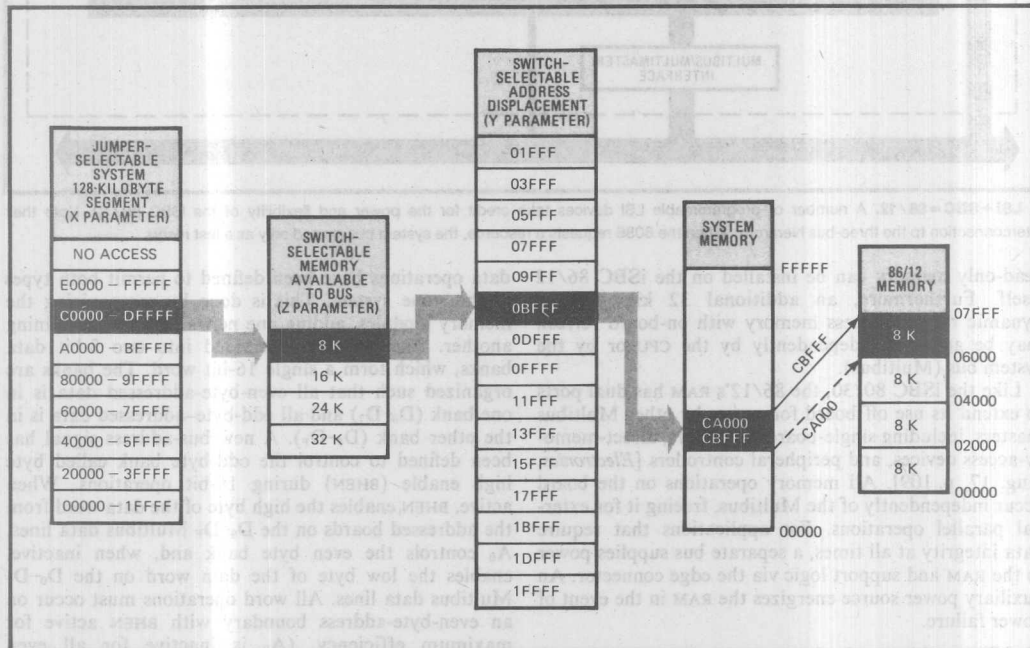
The iSBC 86/12 owes much of its flexibility to programmable large-scale integrated devices. An 8255A peripheral interface chip provides 24 programmable I/O lines that may be tailored to the customer's needs by simply programming the device for input, output, or bidirectional modes with or without handshaking abilities. In conjunction with the 8255A's configuration the user may then select appropriate line drivers and terminators for the I/O lines that can be inserted into sockets on the iSBC 86/12 board.

An 8251A universal synchronous/asynchronous receiver/transmitter is included to provide an RS-232-C interface for serial communication with other computers, RS-232-C-type peripherals (cassette tape, modems, etc.) or cathode-ray-tube terminals. The 8251A enables the user to customize the communication link. Synchronous/asyn-

chronous mode, data format, control character format, parity and baud rates from 75 to 38.4 kilobauds are all under program control.

For system timing functions an 8253 programmable interval timer provides two programmable timers, each of which may be used as a square-wave generator, retriggerable one-shot multivibrator or as an event counter.

The interrupt structure of the iSBC 86/12 encompasses nine levels with vectored priority. Eight of these levels are handled by an 8259A programmable interrupt controller chip, which may be configured for different priority processing modes in accordance with the application. One nonmaskable interrupt is available to immediately alert the CPU to catastrophes like a power failure, in which case the CPU can branch to an appropriate routine in memory to effect an orderly system shut-down.



3. RAM, please. The 8086's view of on-board memory is fixed from zero to 07FFFH. When an outside master accesses this space, the DP controller performs the translation. Here, locations 06000H to 07FFFH are available to another master by addressing CA000H to CBFFFH.

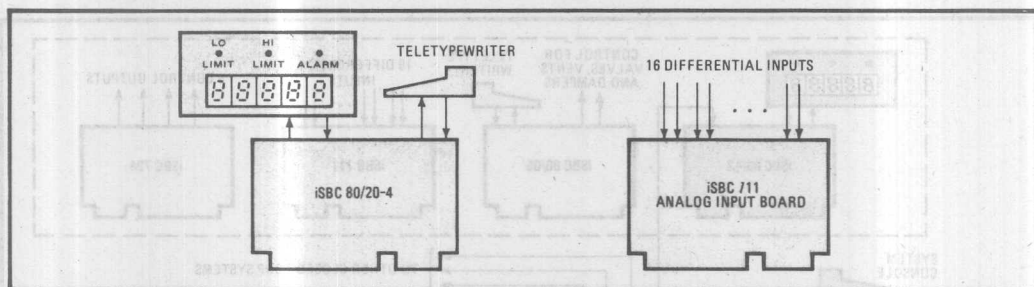
Since all 8-bit accesses via Multibus are done on the lower byte of the data word, the iSBC 86/12 can access 8-bit memory or I/O devices from the system bus. This makes the iSBC 86/12 compatible with all iSBC 80 Multibus modules.

More interrupts, too

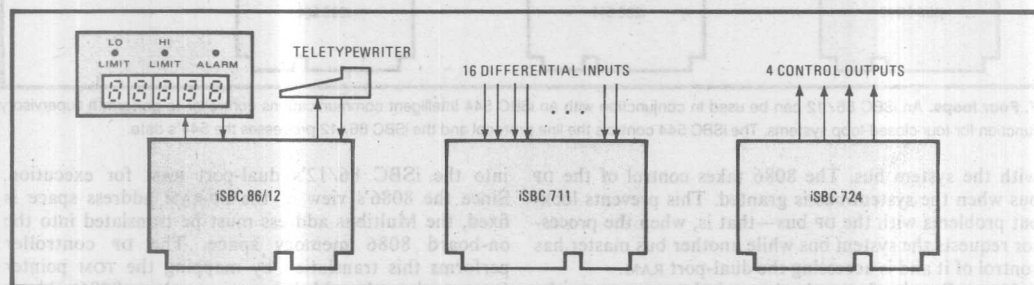
The iSBC 86/12 expands the previous Multibus definition of interrupts by creating two distinct types: non-bus-vectored (NBVI) and bus-vectored (BVI) interrupts. Each Multibus interrupt line can be individually defined

through software to be a BVI or NBVI. Using BVIS, the interrupt capability of a Multibus system can be increased to 64 bus-vectored-priority interrupts.

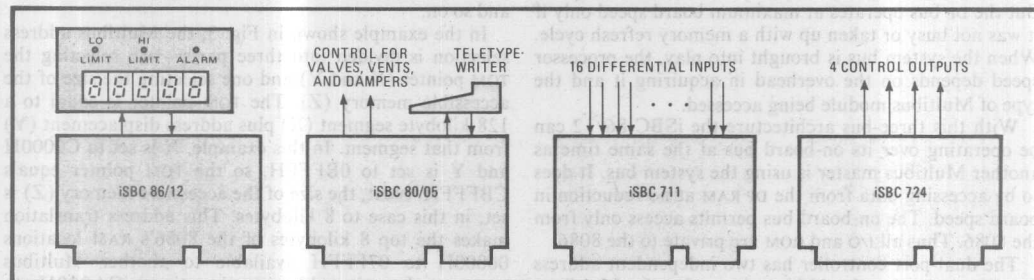
Using NBVIS, a slave module activates an interrupt line and the interrupted bus master generates its own restart address to service that interrupt. The Multibus address or data lines are not used. A BVI uses the Multibus address and data lines to communicate with the interrupting slave. When the slave module generates an interrupt, the bus master requires that module to generate the restart address. One additional command signal is



4. Open loop. Shown above is a simple alarm and monitoring system. The iSBC 711 analog-input board samples 16 differential inputs and the 8-bit iSBC 80/20 compares the inputs to the high and low limits. An alarm condition illuminates an LED and gets logged on a teletypewriter.



5. Closed loop. Suppose the system in Fig. 4 needs to be upgraded to handle a closed-loop system. For this application an iSBC 86/12 replaces the 80/20-04 to cope with the higher processing. The output control variables are handled by an iSBC 724 analog-output board.



6. Multi/master. To enhance the control system in Fig. 5, add a dedicated CPU to control valves, vents, and dampers that, in turn, affect pressure and flow parameters in the system. This has been done by adding an iSBC 80/05 in a Multibus/multimaster arrangement.

defined—interrupt acknowledge (INTA)—to request the restart address from the slave module.

The iSBC 86/12 board architecture, like that of the 8-bit iSBC 80/30, is organized around a three-bus hierarchy: an on-board bus, a dual-port bus and a system bus (Multibus). All three buses have been expanded over their 80/30 counterparts to incorporate 20 address lines and 16 data lines.

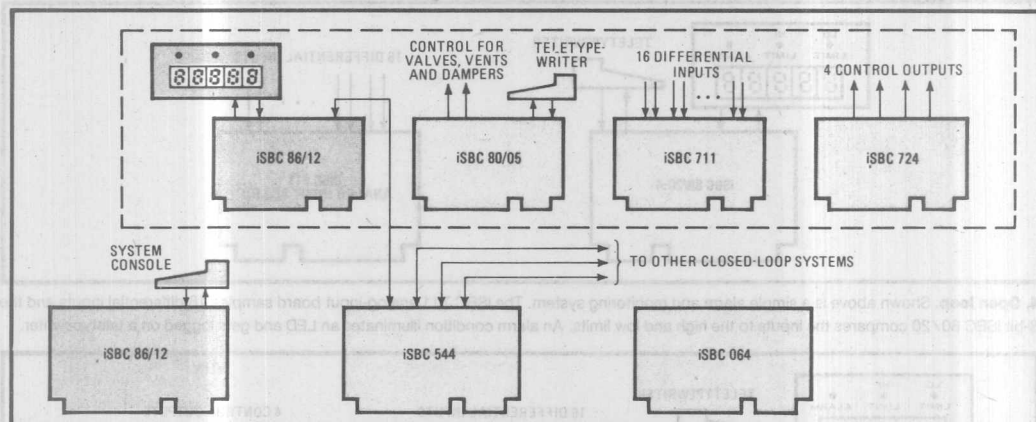
The iSBC 86/12 architecture

The on-board bus links the 8086, all the I/O peripherals, and the read-only memory. Next in the hierarchy is the dual-port bus, which connects to the DP controller, 32 kilobytes of dynamic RAM, and the dynamic RAM controller. Finally, the system bus permits expansion of system resources through Multibus modules (Fig. 2).

The bus protocol of the iSBC 86/12 dictates that each

of the three buses communicate with an adjacent bus or operate independently. When the CPU makes a request for a resource, the on-board and dual-port buses simultaneously determine if their hardware can fulfill the request. If the on-board bus is able to acknowledge the request, it does so and the DP bus is not disturbed. (The DP bus is not interrupted to determine whether it can acknowledge the request.) The 8086 always controls the on-board bus, and requested operations can be completed without delay. If the DP bus is needed, it is requested and the dual-port controller grants the use of the bus to the processor. Thereafter, the dynamic-RAM controller completes the operation and generates an acknowledge.

If neither the on-board nor the DP bus can satisfy the request, the CPU asks for the system bus. The 8086 must use the on-board and dual-port buses to communicate



7. Four loops. An iSBC 86/12 can be used in conjunction with an iSBC 544 intelligent communications controller to perform a supervisory function for four closed-loop systems. The iSBC 544 controls the line protocol and the iSBC 86/12 processes the 544's data.

with the system bus. The 8086 takes control of the DP bus when the system bus is granted. This prevents lock-out problems with the DP bus—that is, when the processor requests the system bus while another bus master has control of it and is accessing the dual-port RAM.

Naturally, the fewer the buses it has to access, the faster the iSBC 86/12 completes a transaction. The on-board bus always operates at maximum board speed. But the DP bus operates at maximum board speed only if it was not busy or taken up with a memory refresh cycle. When the system bus is brought into play, the processor speed depends on the overhead in acquiring it and the type of Multibus module being accessed.

With this three-bus architecture the iSBC 86/12 can be operating over its on-board bus at the same time as another Multibus master is using the system bus. It does so by accessing data from the DP RAM at no reduction in board speed. The on-board bus permits access only from the 8086. Thus all I/O and ROM are private to the 8086.

The dual-port controller has two independent address decoders—one for the 8086 and one for the Multibus. The 8086 decoder fixes the 8086's RAM addresses from hexadecimal 00000 to 07FFF using a fusible-link programmable ROM. The Multibus decoder allows the user to select any address range for the on-board RAM by specifying two parameters—a top-of-memory pointer and the size of the accessible memory. The TOM pointer (as seen by another Multibus master) can be set to any 8-kilobyte boundary in the 1-megabyte memory space. The amount of memory on the iSBC 86/12 accessible by another master can be set to 8, 16, 24, or 32 kilobytes (or no access) with an on-board jumper. For example, fixing the accessible memory size to 24 kilobytes provides the 8086 with 8 kilobytes of RAM that only it can access. This private area can be used for the processor's stack, interrupt jump table and other special system parameters that are generally protected from other Multibus masters. The only addressing restriction is that the memory block accessible to the Multibus cannot cross a 128-kilobyte boundary.

Suppose a Multibus master wants to load a program

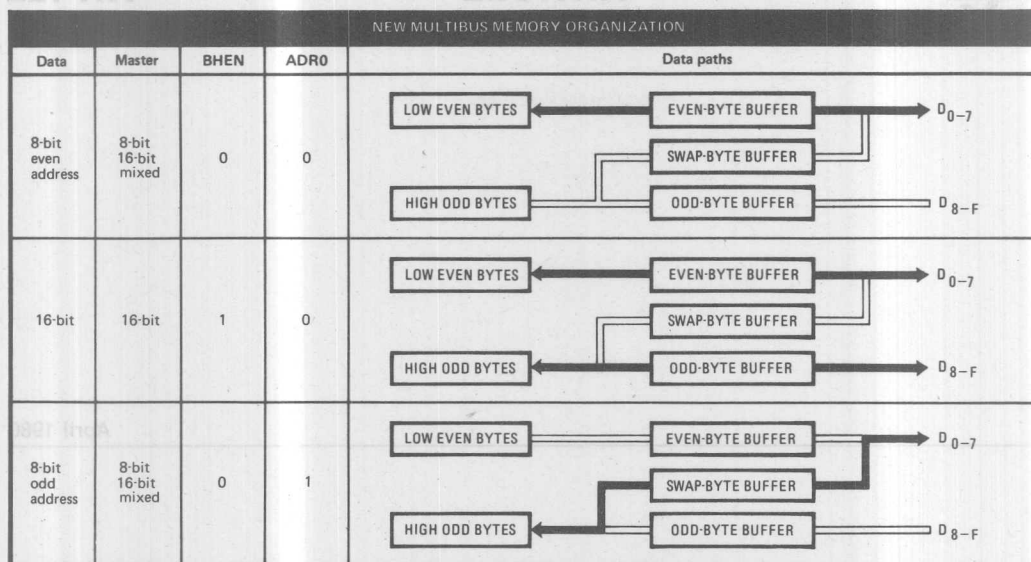
into the iSBC 86/12's dual-port RAM for execution. Since the 8086's view of the DP-RAM address space is fixed, the Multibus address must be translated into the on-board 8086 memory space. The DP controller performs this translation by mapping the TOM pointer (as seen by other Multibus masters) to 8086-address 07FFFFH, the top of the 8086's on-board RAM. Pointer-1 is mapped to the top of 8086 on-board RAM-1, and so on.

In the example shown in Fig. 3, the Multibus address selection is divided into three parts—two selecting the TOM pointer (X and Y) and one selecting the size of the accessible memory (Z). The TOM pointer is equal to a 128-kilobyte segment (X) plus address displacement (Y) from that segment. In this example, X is set to C0000H and Y is set to 0BFFFFH, so the TOM pointer equals CBFFFFH. Next, the size of the accessible memory (Z) is set, in this case to 8 kilobytes. This address translation makes the top 8 kilobytes of the 8086's RAM locations 06000H to 07FFFFH available to another Multibus master when it addresses locations CA000H to CBFFFFH. The 8086 still has 24 kilobytes (00000H to 05FFFFH) of private memory.

Multiprocessing schemes

In multiprocessing systems, a master must be able to access the system without another master obtaining the bus. The iSBC 86/12 incorporates bus-arbitration logic to effect these transactions. Since the system bus is only requested when a system resource is needed, the iSBC 86/12 can perform true parallel processing with other iSBC 80 or 86 masters.

A typical example is the use of a common memory location that contains the status byte (busy/not busy) of a floppy-disk controller. When the floppy disk is needed, the master must first read the location and, if not busy, write the status word without another master obtaining the bus (to use the floppy disk). A bus-lock function on the iSBC 86, once enabled, allows the iSBC 86 to maintain control of the system bus until the lock is disabled by program control. This bus-lock function may



be activated in one of two ways—by an output bit from the resident 8255A peripheral-interface chip or by a software prefix on any 8086 instruction. The iSBC 86 can perform the test and set function by exchanging the accumulator with the memory location, preceding the instruction by a lock prefix. For example, the status word is read into the accumulator and, without another intervening bus cycle, a busy status is written. The accumulator is then tested: if busy—try again (writing a busy does not destroy status as it was already busy); if not busy, the floppy disk is now under the master's control and the status location is set to busy.

The iSBC 86/12: a design tool

For system debugging and full-speed execution, the iSBC 86/12 can be linked to the Inteltec microcomputer development system. Programs generated on the Inteltec system can be downloaded into the iSBC 86/12 RAM via cables. Through a virtual-terminal capability, the Inteltec console can directly access an iSBC-resident monitor, which provides commands for software debug. Once the debugging cycle is completed, the user has the option of uploading the software back to the Inteltec for storage on diskettes.

The Multibus and form-factor compatibility of the 8-bit iSBC 80 and 16-bit iSBC 86 single-board computers provide a degree of design flexibility previously unobtainable. Initial design problems can be solved with low-cost 8-bit hardware. As product requirements evolve, 16-bit performance can be added. Eventually, 8- and 16-bit multiprocessor solutions can be conveniently implemented.

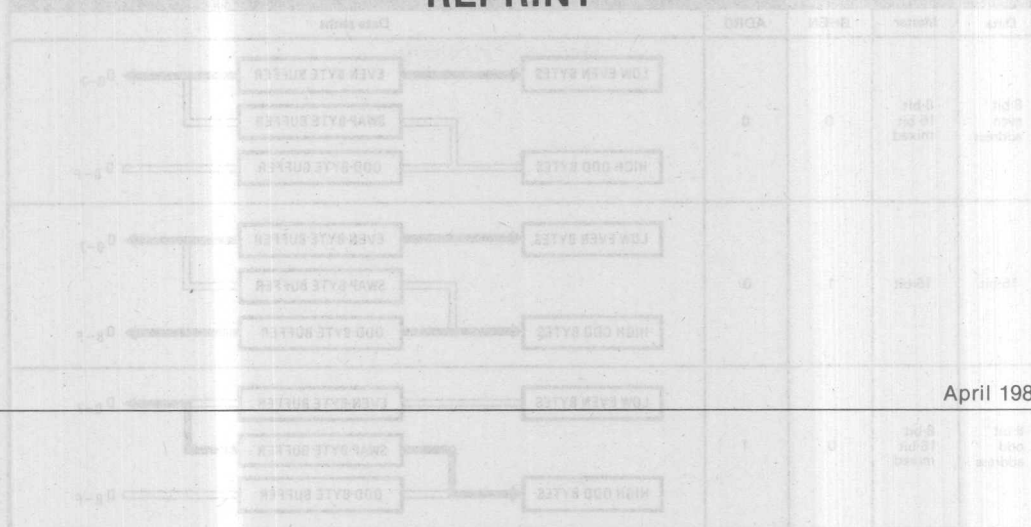
Consider the application shown in Fig. 4, an alarm and monitoring system in a typical process plant. Sixteen differential inputs from pressure and flow transducers are sampled once every second, then compared to high and low limits previously entered through thumbwheel

switches. The iSBC 711 analog-input board takes care of sampling the inputs and the 8-bit iSBC 80/20-4 compares the data to the high and low limits. Whenever these limits are exceeded, an alarm LED lights up and the alarm condition is logged on the system teletypewriter along with input identification, high limit, low limit and sampled value.

Closed loops

Instead of an open-loop system, suppose the design must be enhanced to control four output variables—thereby making it a closed-loop system. The sampling rate must be increased to once every third of a second and more processing will be required to run through the control algorithm and output the control-loop data. For this application, and iSBC 86/12 replaces the 80/20-4 to handle the higher processing requirements. An iSBC 724 analog-output board is also added to provide the four output-control variables (see Fig. 5). Carrying this example one step further, one may want to dedicate another processor to controlling valves, vents, and dampers that in turn affect pressure and flow parameters in the system. This can be done by adding an iSBC 80/05 in a multimaster arrangement as shown in Fig. 6.

Finally, an iSBC 86/12 can be used with an iSBC 544 intelligent communication-controller to supervise four closed-loop systems of the type shown in Fig. 6. The 86/12 of each system interfaces with the supervisory system via its serial interfaces, which are connected to the iSBC 544's serial ports (see Fig. 7). The iSBC 544 performs the control functions associated with the line protocol. The supervisory iSBC 86/12 can access the iSBC 544's dual-port memory and can perform further processing of the data received from the four closed-loop systems. In this configuration large amounts of memory may be required; since the iSBC 86/12 can address up to 1 megabyte, this presents no problem. □



April 1980

World of MULTIMODULE Intel's Computers

New Family Plans Extended by In- crease in Board C

A New Board Provides Single

A New Family of MULTIMODULE™ Boards Extends the Solutions Provided by Intel's Single Board Computers

Preview Magazine, March/April 1980

Preview Magazine, March/April 1980

FEATURE

A new family of MULTIMODULE™ boards extends the solutions provided by Intel's single board computers

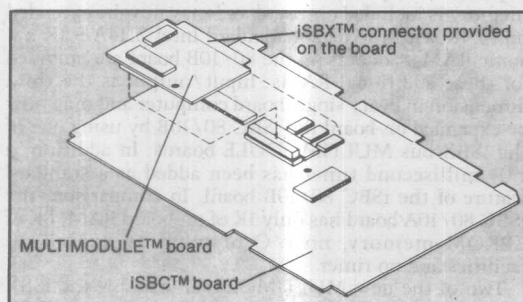
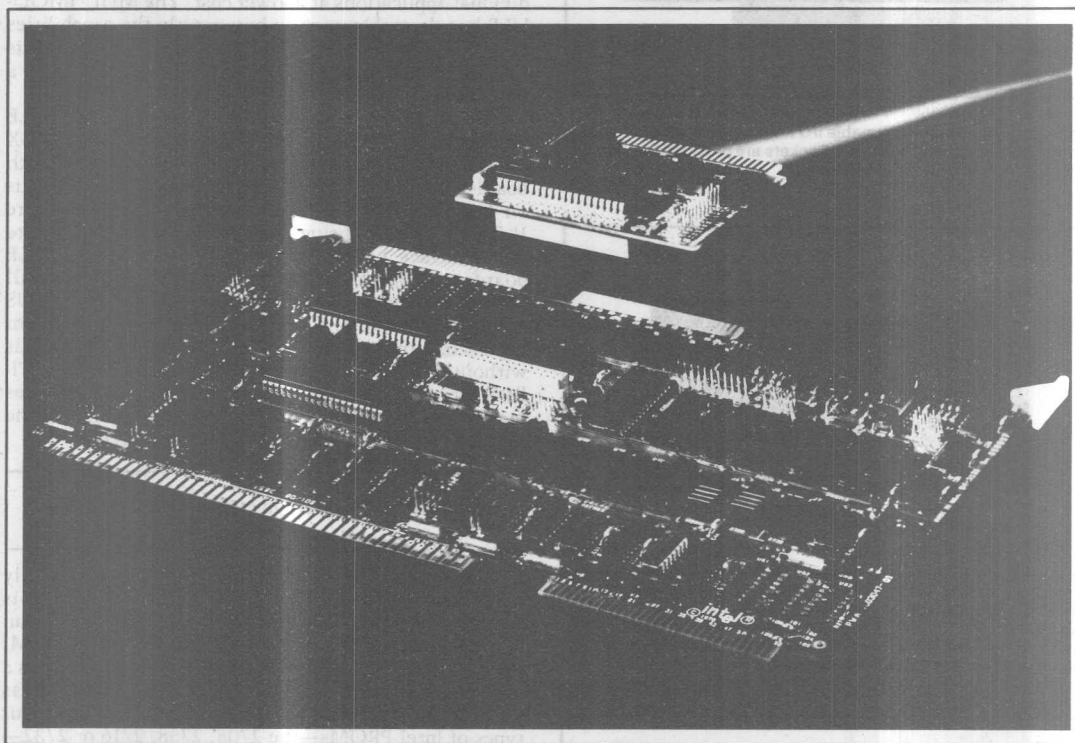


Figure 1. The iSBX 350™ Programmable I/O MULTIMODULE plug-in, shown here, connects directly on one end into the special iSBX bus connector and screws onto the edge of the single board computer on the other end.

A complete new family of products, called MULTIMODULE boards, has been announced by Intel Corporation. The new MULTIMODULE boards designed to extend the functional capabilities of single board computers at much lower cost than has been previously possible. Intel is supporting the MULTIMODULE concept with a new bus standard—the iSBX bus. The iSBX bus will now be designed into a new generation of single board computers to achieve compatibility with the emerging iSBX bus compatible MULTIMODULE product line. Users of Intel's single board computers can incrementally expand system resources by adding small (2.85" x 3.7") iSBX MULTIMODULE boards which plug directly into iSBC boards.

Three new iSBX bus MULTIMODULE plug-ins have been introduced—modules for parallel I/O, serial I/O,

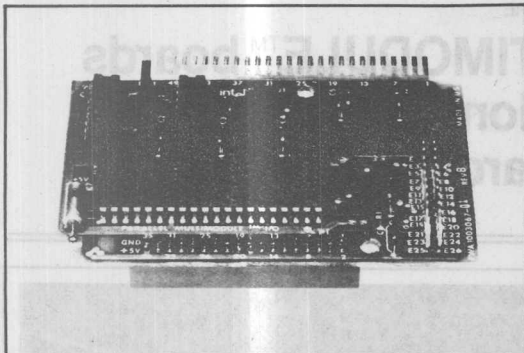


Figure 2. The iSBX 350™ Programmable I/O MULTIMODULE™ board provides 24 programmable I/O lines using the 8255A-5 programmable peripheral interface. Sockets are provided for interchangeable line drivers/terminators.

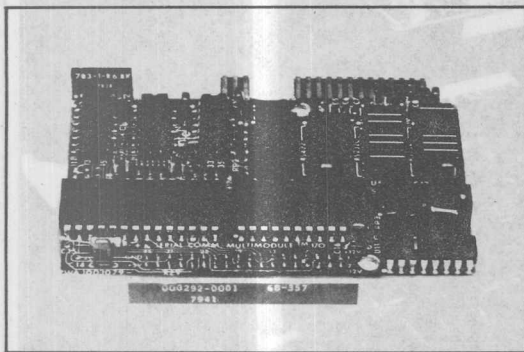


Figure 3. The iSBX 351™ Serial I/O MULTIMODULE board extends the serial communications capability of a single board computer via an Intel® 8251A USART. It includes an on-board programmable baud rate generator, two programmable 16-bit BCD/binary timers, and RS232C or RS422/449 interfacing.

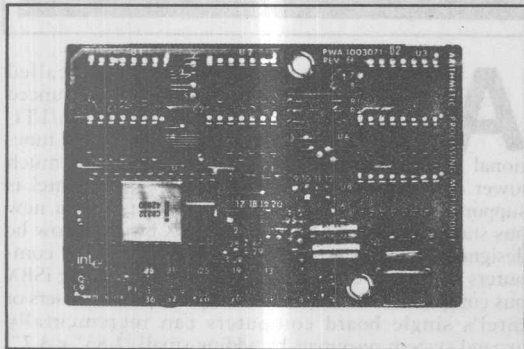


Figure 4. The iSBX 332™ Floating Point Math MULTIMODULE board uses an Intel® 8232 Floating Point Processor and is compatible with the new proposed IEEE format. It provides users with both single-precision (32-bit) and double-precision (64-bit) arithmetic functions.

and floating-point math. The showcase for the MULTIMODULE family is the iSBC 80/10B board—the first iSBX bus compatible single board computer.

MULTIMODULE™ boards allow users to tailor board level products

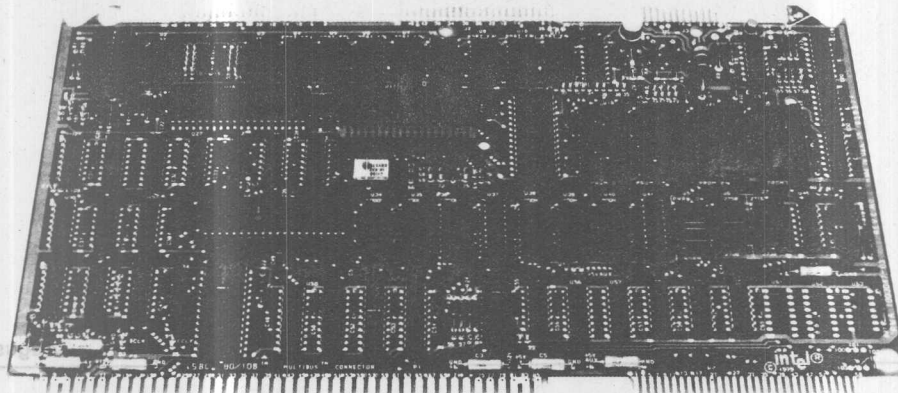
Customers can choose MULTIMODULE boards to precisely configure single board computers for their individual applications at a lower cost. The MULTIMODULE boards enable users to buy exactly the capabilities that they require for their iSBC-based systems. This can, of course, keep system size and system cost at a minimum.

Any of the three new iSBX bus MULTIMODULE products—the iSBX 350 Programmable I/O, the iSBX 351 Serial I/O, and the iSBX 332 Floating Point Math board—plug into a special interface called the iSBX bus on the iSBC 80/10B single board computer, (see Figure 1). The iSBX bus is being introduced by Intel to facilitate the interface between the iSBX MULTIMODULE boards and the iSBC products. The iSBX bus will become a standard, similar to the standard MULTIBUS system architecture. The new iSBX bus allows system expansion through the new MULTIMODULE boards without making any demands on the system's MULTIBUS interface. As a result, the system design achieves maximum on-board performance while freeing up the MULTIBUS interface for other system activities.

iSBC 80/10B™ board is customer-expandable

The new iSBC 80/10B single board computer is fully compatible with the popular iSBC 80/10A, but it is customer-expandable in "all directions." This board can be expanded in three dimensions to tailor EPROM, RAM, and I/O needs directly on the board. This gives a user a built-in adaptability on one board that was previously unattainable. First, the user can choose four types of Intel PROMs—the 2708, 2758, 2716 or 2732—expandable up to 16K bytes. Second, 1K of RAM memory is included on-board and may now be extended up to 4K with the use of standard Intel 2114A-5 1K x 4 static RAMs. Sockets on the 80/10B board are provided for these additional RAMs. Input/output is the third dimension in every single board computer and may now be expanded on-board the iSBC 80/10B by using one of the iSBX bus MULTIMODULE boards. In addition, a 1.04 millisecond timer has been added as a standard feature of the iSBC 80/10B board. In comparison, the iSBC 80/10A board has only 1K of on-board RAM, 8K of EPROM memory, no I/O or memory expansion facilities and no timer.

Two of the new MULTIMODULE boards—the iSBX 350 and iSBX 351—provide expansion to the I/O already on the iSBC 80/10B board. The iSBX 350 Programmable I/O MULTIMODULE (see Figure 2) provides 24 I/O lines with sockets for interchangeable line



iSBC 80/10B™ Single Board Computer

Here's what iSBX™ bus MULTIMODULES™ provide the user:

- The ability to incrementally expand the iSBC Single Board Computer, via iSBX bus, allowing the user to add only the function his application requires.
- The on-board addition of totally new capabilities (mathematics processing and the like) to single board computers.
- Maximum performance by reducing traffic required for standard expansion boards on the industry standard MULTIBUS interface.
- Compatibility with future 8-bit and 16-bit single board computers from Intel
- A high-reliability, 36-pin iSBX 960-5 male connector for customer design

drivers and terminators. The iSBX 351 Serial I/O MULTIMODULE (see Figure 3) provides programmable, synchronous/asynchronous RS232C or RS422/449 compatible serial expansion with fully software selectable baud rate generation. Additionally, two programmable 16-bit BCD and binary timers are available, allowing iSBC 80/10B users to implement programmable timing functions directly on the board.

A third MULTIMODULE board which users may select, provides additional on-board mathematics capability to their single board computer. The iSBX 332 Floating Point Math MULTIMODULE board (see Figure

board computer and full software reset control.

User may easily implement their more specialized requirements on the new iSBX bus through a readily available 36-pin male connector, the iSBX 960-5 (packaged with 5 connectors). In addition, Intel is providing an iSBX bus specification describing the electrical and mechanical parameters of the interface. The iSBX 960-5 male connectors mate directly to the female iSBX bus connector on the iSBC 80/10B single board computer. The connector, together with the iSBX bus specification, allows system designers to take full advantage of MULTIMODULE benefits.

New MULTIMODULE boards will be introduced—as will new generation iSBX bus compatible iSBC single board computers—by Intel throughout the remainder of 1980 and into the future. MULTIMODULE boards and the iSBX bus represent a major commitment by Intel for the future and offer system designers new options to complement single board computers and MULTIBUS expansion.

The immediately available MULTIMODULE family—with the iSBC 80/10B single board computer—provides great flexibility and savings for users. The options now exist to expand a system in small increments with MULTIMODULE boards, or in large increments with MULTIBUS boards. And, with the three dimensional expansion capability of the iSBC 80/10B board, the user may configure his entire system on-board to achieve a single board, low cost solution.

New iSBX 960-5™ MULTIMODULE™ connectors are available off-the-shelf

4) is compatible with the proposed IEEE format standard. It offers single-precision (32-bit) and double-precision (64-bit) arithmetic functions including Add, Subtract, Multiply and Divide. It also includes end-of-operation and error interrupts to its host iSBC single

PRICE: Call your local Intel sales office or distributor for the latest prices on all of the MULTIMODULE products

AVAILABILITY: Now

LITERATURE: iSBC 80/10B Single Board Computer N/C
Data Sheet
iSBX 350 MULTIMODULE Data Sheet N/C
iSBX 351 MULTIMODULE Data Sheet N/C
iSBX 332 MULTIMODULE Data Sheet N/C

August 1980

Special-Function Modules Ride on Computer Board

Gary Sawyer, Jim Johnson, Dave Jurasek, and Steve Kassel
 Electronics, April 10, 1980

Gary Sawyer, Jim Johnson,
Electronics, April 10, 1980

Special-function modules ride on computer board

Smaller cards donate floating-point processing or added serial and parallel I/O to primary single-board computer; memory is extensible on the main card

by Gary Sawyer, Jim Johnson, David Jurasek, and Steve Kassel, Intel Corp., Hillsboro, Ore.

□ In the design of board-level computers, two basic methods coexist. One is to pack each card with integrated circuits to the limits of its capacity, and the other is to distribute the computer functions among other boards occupying additional card slots.

Both approaches have their advantages. The single powerful module conserves space and expensive connectors, while the decentralized boards allow the user to pick and choose functions—and add them incrementally—although the expense of one board might spell overkill for one particular application.

A new concept in single-board computer architecture strikes a neat compromise between both camps. Rather than cram more chips on an already overstuffed board, the idea is simply to provide it with a connector for plugging in smaller modules having limited functions for specialized applications.

Best of both worlds

This is the idea behind iSBX Multimodule boards, which cost from \$155 to \$450 apiece. Plugging into a primary processor card, 10.5-square-inch boards with various types of memory or input and output functions provide the larger single-board computer with more versatility. Linking the base board and these Multimodule boards is a new 36-line bus called the iSBX bus, for single-board expansion. This interface is destined to match the popularity of the main board's Multibus interface connector (Fig. 1).

The iSBX bus is derived directly from the on-board microprocessor system bus and, as such, an iSBX-compatible board becomes an integral element of the single-board computer. The physical interface uses a unique connector designed specifically for the iSBX bus. The bus is brought to a female connector on the single-board computer; its male equivalent is resident on the iSBX board (Fig. 2).

The iSBC 80/10B board in Figs. 1 and 2 is the first single-board computer to be compatible with the iSBX bus. Upwardly compatible with its predecessor, the iSBC 80/10A, the iSBC 80/10B is functionally equivalent but offers significant enhancements.

The iSBC 80/10B board offers direct functional expansion in three dimensions—not only read-only memory (as in the iSBC 80/10A), but also static random access memory and input and output, as facilitated by

the Multimodule boards. One kilobyte of static RAM is provided along with sockets for expansion in increments of 1-K bytes to 4-K bytes using standard 2114A-5 memories. Read-only memory may be expanded with standard ultraviolet-light-erasable and mask-programmable types to 16-K bytes.

The iSBC 80/10B also features an on-board 1.04-millisecond timer with ongoing clocking that users may optionally configure for microprocessor interrupts. In addition, power-fail control is provided for the 2114A-5 static RAMs, enabling the user to add battery backup if the memory contents must be preserved.

Three Multimodule boards

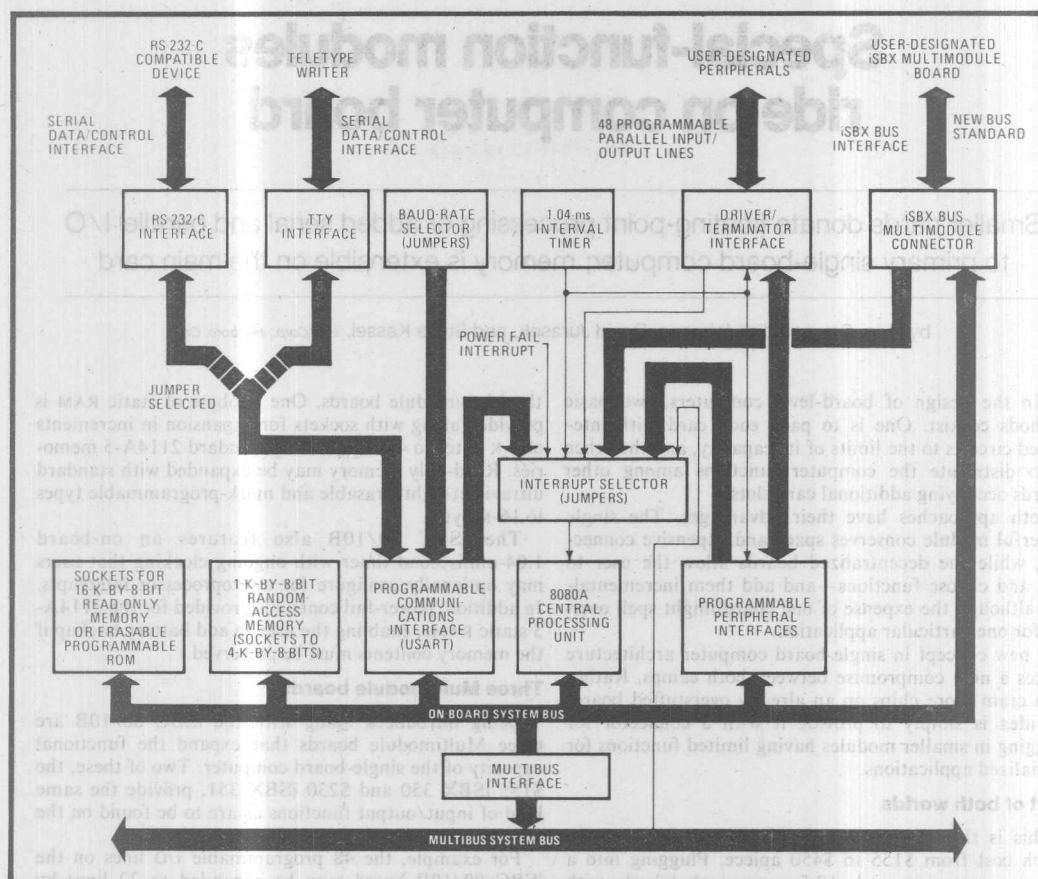
Being introduced along with the iSBC 80/10B are three Multimodule boards that expand the functional capacity of the single-board computer. Two of these, the \$155 iSBX 350 and \$230 iSBX 351, provide the same kind of input/output functions as are to be found on the processor board, only more of them.

For example, the 48 programmable I/O lines on the iSBC 80/10B board may be expanded to 72 lines by simply plugging in the iSBX 350 module—a 50% increase. Serial I/O is similarly expanded with the iSBX 351 module, which provides a programmable universal synchronous-asynchronous receiver/transmitter, or Usart (an 8251A), for compatibility with the RS-232-C and RS-449/422 interfaces. The iSBX 351 module further offers software-selectable baud rates and two programmable 16-bit binary or binary-coded decimal timers.

The third Multimodule board adds otherwise unavailable high-speed math capabilities to the iSBC 80/10B board. The \$450 iSBX 332 board uses the 8232 floating-point processor for arithmetic compatible with the standard currently being proposed by the Institute of Electrical and Electronics Engineers.

Many applications require a custom design. To complement the standard family of Multimodule boards, the iSBX 960-5 is provided. This includes five male iSBX connectors, and a full bus specification is available for custom interfacing by the user. This combination permits a user to satisfy his or her requirements for specialized I/O interfaces with the Multimodule concept.

The Multimodule concept can be divided into two logical elements: base boards and Multimodule boards.



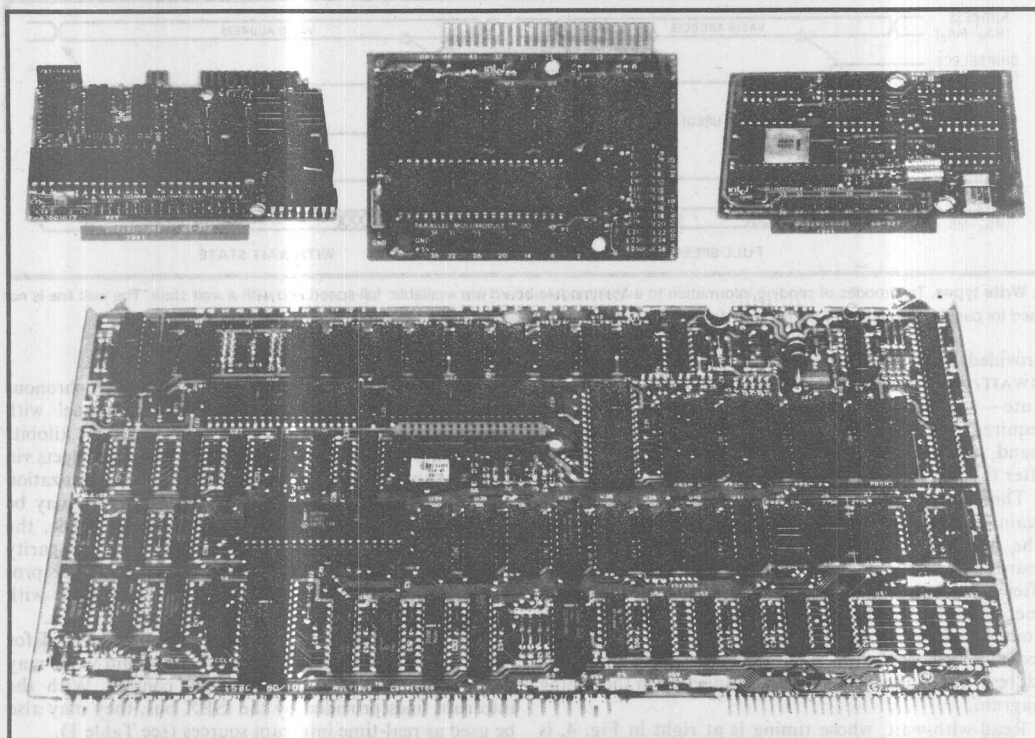
1. Distributed. The first processor card to receive the iSBX bus connector is the iSBC 80/10B, a follow-on to the iSBC 80/10A. The off-board system bus is the Multibus, which interfaces to the on-board system bus. This, in turn, connects to the Multimodule board connector.

The base board is the master of the system in that it controls communication between the base's microprocessor and the Multimodule board's port. Though the first base board is a single-board computer—the iSBC 80/10B—Multibus-compatible slaves and intelligent I/O boards will also incorporate iSBX bus interfaces. The Multimodule board is a slave of the system in that it carries out I/O commands from the base board.

The iSBX interface

The iSBX bus specification includes both electrical and mechanical characteristics. The mechanical interface is convenient and rugged; the Multimodule board is mounted to the base board in two places, at the top with a screw and at the bottom by the iSBX bus connector. The connector is extremely reliable. It has gold-plated phosphor-bronze contacts, it is keyed to assure proper orientation, and a shroud protects its pins during handling. The connector also incorporates interlocking tabs to ensure a solid mechanical interface.

Electrically, the iSBX bus interface lines can be



2. Three to one. Below is the iSBC 80/10B main processor board, and above it are the three new Multimodule boards. They are, from left to right, the iSBX 351 serial I/O module, the iSBX 350 parallel I/O module, and the iSBX 332 floating-point mathematics Multimodule board.

base-board processor into a wait state, allowing the Multimodule board extra time to perform a requested operation, if necessary. $MWAIT/$ is generated from address and chip-select information only. $MPST$ is tied to ground on the Multimodule board to inform the base board that a Multimodule board has been installed.

The second class of iSBX bus lines includes the address lines (MA_0 - MA_2) and the chip-select lines (MCS_0 and MCS_1). The base board decodes I/O addresses to generate the chip-select signals for the Multimodule boards. In so doing, it normally decodes all but the three lowest-order addresses (MA_0 - MA_2). A base board normally reserves two blocks of eight I/O ports for each iSBX bus connector provided.

Defining the lines

Eight bidirectional data lines (MD_0 - MD_7 , active high) carry information to and from the Multimodule ports. MD_0 is the least significant bit. The two active high interrupt lines from the Multimodule board, $MINTR_0$ and $MINTR_1$, make interrupt requests to the base board.

Two optional lines, OPT_0 and OPT_1 , are connected to wire-wrapped posts on both the base and Multimodule boards. They may serve either as additional interrupts from the Multimodule board or as special signals from the base board.

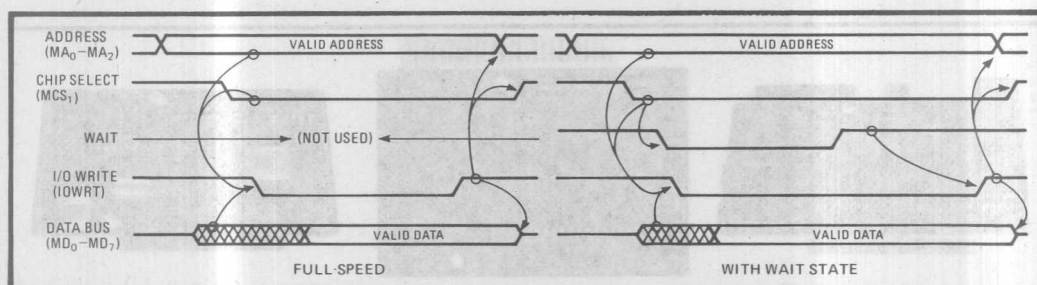
Finally, all base boards provide +5 and ± 12 volts to the Multimodule boards. These power lines complete the six iSBX bus classes.

The primary function of the iSBX bus is to provide a path for I/O-mapped data between base board and Multimodule board. This happens when the base board performs an I/O-read or I/O-write operation. There are two types of I/O-write operations, and the Multimodule board determines which is performed.

Data transfers

The first is a full-speed I/O write (Fig. 3). The base board generates a valid I/O address and chip-select and activates the $IOWRT/$ line after the set-up times are met. The $IOWRT/$ line will remain active for a minimum of 300 nanoseconds and the data will be valid for a minimum of 250 ns before $IOWRT/$ is removed. The base board then removes the data, address, and chip-select signals after the hold times shown in the timing diagram.

The alternative I/O write is a write-with-wait, used by Multimodule boards that cannot write into an I/O port at full speed. Again, the base board generates a valid address and chip-select. The Multimodule board activates the $MWAIT/$ signal based on address and chip-select information. This will remove the ready condition from the processor, causing it to go into a wait state after the write command has been activated and valid data



3. Write types. Two modes of sending information to a Multimodule board are available: full-speed and with a wait state. The wait line is not used for peripherals that meet the full-speed specifications. The wait signal extends the time for which data remains valid.

provided. The Multimodule board will remove the MWAIT/ signal—allowing the processor to leave its wait state—when it has satisfied the write-pulse-width requirement. The base board removes the write command, then the data, address, and chip-select signals, after the hold times are met.

There are two types of I/O-read operations as well, and again they are determined by the Multimodule board. The first is a full-speed I/O read (Fig. 4). The base board generates a valid I/O address and chip-select and, after the set-up timings are met, it activates the IORD/ line. The Multimodule board must generate valid data from the addressed I/O port in less than 250 ns. The base board reads the data and removes the command, address, and chip-select signals as indicated in the timing diagram.

Read-with-wait, whose timing is at right in Fig. 4, is used by Multimodule boards that cannot perform a read operation under the full-speed specifications. The base board generates a valid address and chip-select, just as with a full-speed read. However, the Multimodule board now activates the MWAIT/ signal, which in turn removes the ready input to the base's processor, putting it into a wait state. The processor activates the IORD/ signal before going into a wait state.

The Multimodule board will remove the MWAIT/ signal when valid data can be read from the data bus. After reading the data, the base board removes the command, address, and chip-select signals.

The iSBX 351 serial I/O board is a good example of how easily large-scale integrated circuits may be interfaced by the iSBX bus. It presents the iSBC 80/10B

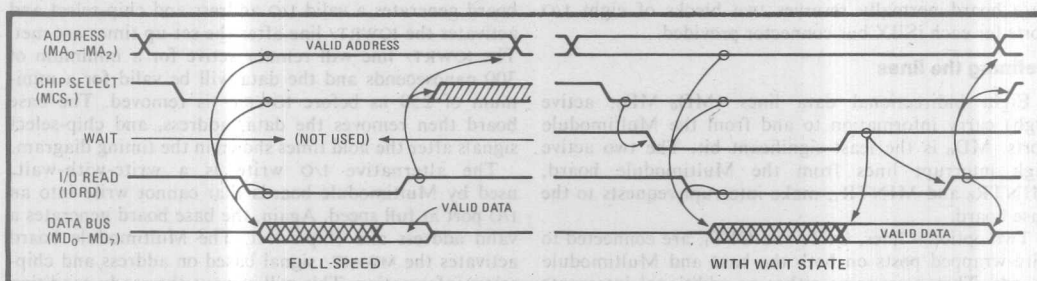
board with a second serial port.

The iSBC 351 board (Fig. 5) provides a synchronous or asynchronous serial communications channel with programmable format and baud rates up to 64 kilobits per second. In the synchronous mode, the user selects via software the number and format of the synchronization characters and the number of data bits. Parity may be even, odd, or disabled. In the asynchronous mode, the number of data bits and stop bits, as well as parity generation and detection, may be specified under program control. The added channel is compatible with either the RS-232-C or RS-422/449 interface.

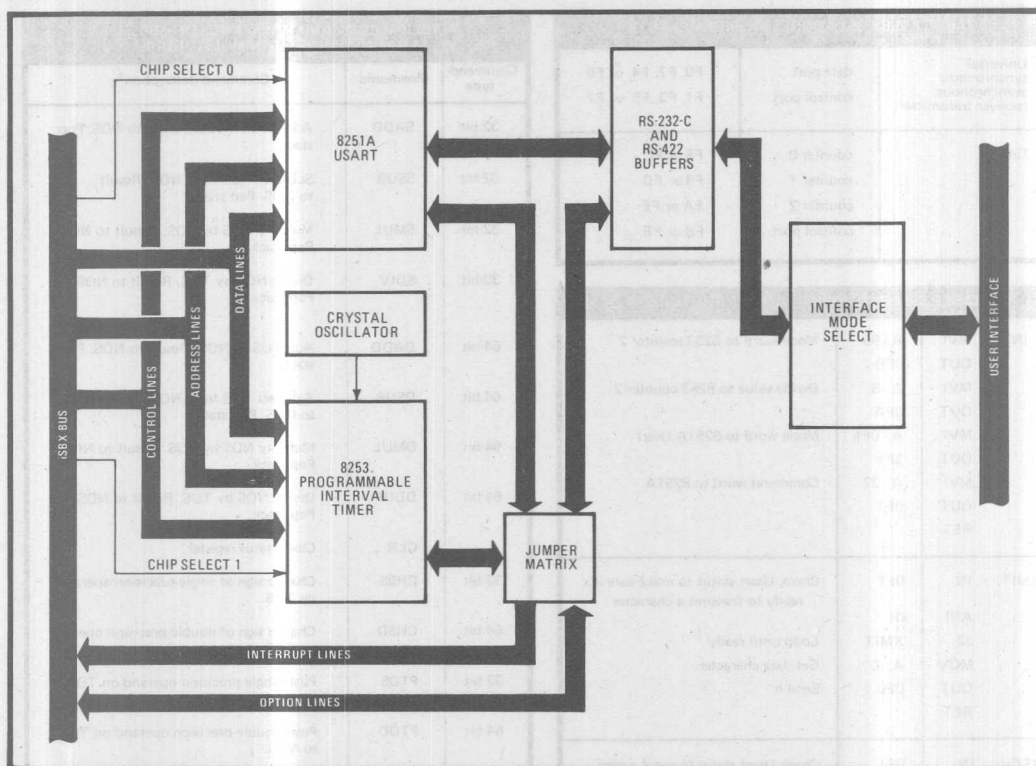
Two additional 16-bit counters are on the board for other uses. Their mode of operation and count value may be written or read under program control. With the interrupt lines provided by the iSBX bus, they may also be used as real-time interrupt sources (see Table 1).

As stated earlier, an 8251A Usart gives the iSBX 351 module a high-performance communications channel. In addition, an 8253 programmable interval timer (PIT) provides the three counters for clock generation and timing. Note in the block diagram of Fig. 5 that both devices are connected directly to the data, address, and command buses with no buffers. (Each chip, however, has its own chip-select line, preventing data bus contention.) The absence of buffers keeps the parts count down and the speed up.

Also shown in the extra block diagram are two optional lines that may be used as additional interrupt lines or to interface to the additional timer/counters. There are four interrupt sources on the board. Two, from the 8251A, indicate either that a character has been received



4. Read types. As in writing data into a Multimodule board, information is read from it in one of two ways: full speed or read-with-wait. The wait state extends the length of time that data is valid. This is necessary for slower transactions such as analog-to-digital conversions.



5. More serial I/O. The iSBX 351 serial input/output Multimodule board provides the base processor with an additional synchronous or asynchronous communications channel—one that is compatible with either the RS-232-C or the RS-422/449 interface specifications.

for reading or that the transmitter buffer is empty and ready for transmission. Two other interrupts may be generated by the timer/counters. The timers count from an on-board crystal-controlled oscillator.

Compatible with both

The iSBX 351 module uses a unique split-edge connector to provide compatibility with both RS-232-C and RS-449 interfaces. RS-232-C is commonly used to communicate with terminals, modems, and other equipment up to a distance of 50 feet away. RS-422 is a new interface that allows high-speed data transfers of up to 4,000 feet through differential lines that reduce noise such as crosstalk. The iSBX 351 module is the first expansion board to offer both interfaces.

The iSBX 351 module is programmed by a series of I/O-read and -write commands. Table 2 shows the I/O port assignments on the iSBC 80/10B board by way of explaining the code sequences in Table 3 that run on it. The first routine in Table 3, INIT, initializes the 8251A for asynchronous operation and programs the 8253 to generate a baud rate of 9,600. XMIT takes a character from the C register of the 8080A and sends it to the Usart for transmission. RECV gets a character from the Usart and places it in the accumulator. Note that in both data-transfer routines the Usart status register is check-

ed to ensure proper operation.

The iSBX 350 programmable I/O Multimodule board provides 24 general-purpose I/O lines (or three 8-bit ports) via a standard 50-pin edge connector, giving the iSBC 80/10B a total of 72 I/O lines. The 8255A is the only LSI component on the board, and six sockets are provided for line drivers or terminators.

Two bidirectional inverting 4-bit bus transceivers are provided for one of the three ports; sockets for the other two are TTL-compatible, allowing the use of inverting, noninverting, or open-collector drivers. When either of these other two ports is used as an input, the lines may be terminated either with 1-kilohm pullup resistor packs or with 220/330-ohm pullup/pulldown resistors.

TABLE 1: iSBX 351 SERIAL INPUT/OUTPUT BOARD'S BAUD RATES AND INTERVAL TIMES

	Minimum values	Maximum values
Baud generator	18.75 bauds	64 kilobauds (limited by 8251A)
Single timer	1.63 μ s	428 ms
Dual cascaded timers	3.26 μ s	7.8 h

TABLE 2: iSBX 351 ADDRESS ASSIGNMENTS

Universal synchronous/asynchronous receiver-transmitter	data port control port	F0, F2, F4, or F6 F1, F3, F5, or F7
Timer	counter 0 counter 1 counter 2 control port	F8 or FC F9 or FD FA or FE FB or FF

TABLE 3: SERIAL INPUT/OUTPUT ROUTINES

INT:	MVI	A, 96	Mode word to 8253 counter 2
	OUT	0FB	
	MVI	A, 8	Divide value to 8253 counter 2
	OUT	0FA	
	MVI	A, 0FE	Mode word to 8251A Usart
	OUT	0F1	
	MVI	A, 27	Command word to 8251A
	OUT	0F1	
	RET		
XMIT:	IN	0F1	Check Usart status to make sure it's ready to transmit a character
	ANI	01	
	JZ	XMIT	Loop until ready
	MOV	A, C	Get data character
	OUT	0F0	Send it
	RET		
RECV:	IN	0F1	Check Usart status to see if a new character has been received
	ANI	02	
	JZ	RECV	Loop until data is available
	IN	0F1	
	ANI	38	Check framing, overrun, and parity error bits
	JNZ	ERROR	Jump to an error handler if there are any problems
	IN	0F0	Get data
	RET		

The iSBX 350 module supports all three 8255A modes: basic I/O, strobed I/O, and strobed bidirectional bus I/O. Several of the handshaking signals are available as interrupt sources, and an additional external interrupt may be brought in via the edge connector.

Programming this board is as simple as programming the 8255As on the iSBC 80/10B itself. First, a mode word is written to the control port to specify the operational mode for each port. Data transfer may then begin, in the form of I/O-read or -write operations.

The iSBX 332 module is an accurate 32- or 64-bit floating-point processor that performs arithmetic operations in accordance with the proposed IEEE floating-point standard. It uses the 8232 floating-point processor.

The math module uses one data format that has two word lengths of 32 or 64 bits. The board will add, subtract, multiply, and divide for both word lengths.

TABLE 4: COMMAND MNEMONICS OF iSBC 332 FLOATING POINT MATH MULTIMODULE

Command type	Mnemonic	Command description ¹
32-bit	SADD	Add TOS to NOS. Result to NOS. Pop stack.
32-bit	SSUB	Subtract TOS from NOS. Result to NOS. Pop stack.
32-bit	SMUL	Multiply NOS by TOS. Result to NOS. Pop stack.
32-bit	SDIV	Divide NOS by TOS. Result to NOS. Pop stack.
64-bit	DADD	Add TOS to NOS. Result to NOS. Pop stack.
64-bit	DSUB	Subtract TOS from NOS. Result to NOS. Pop stack.
64-bit	DMUL	Multiply NOS by TOS. Result to NOS. Pop stack.
64-bit	DDIV	Divide NOS by TOS. Result to NOS. Pop stack.
—	CLR	Clear status register.
32-bit	CHSS	Change sign of single-precision operand on TOS.
64-bit	CHSD	Change sign of double-precision operand on TOS.
32-bit	PTOS	Push single-precision operand on TOS to NOS.
64-bit	PTOD	Push double-precision operand on TOS to NOS.
32-bit	POPS	Pop single-precision operand from TOS. NOS becomes TOS.
64-bit	POPD	Pop double-precision operand from TOS. NOS becomes TOS.
32-bit	XCHS	Exchange single-precision operands TOS and NOS.

1. abbreviations: NOS = next on stack, TOS = top of stack

Table 4 shows the instruction mnemonics and functions, as well as the positions in the stack (top of stack or next on stack) the operands and results occupy.

The 8232 runs at 4 MHz for maximum throughput. A multiplication of two 32-bit quantities takes about 50 microseconds, excluding data entry and retrieval. In addition, two interrupts signal the base-board processor of completion of an operation or an error.

Floating-point math

The two word lengths of the floating-point standard were chosen for the highest speed and accuracy. If speed is the primary objective, the 32-bit format gives a dynamic range of approximately 10^{-38} to 10^{+38} . If range and accuracy are required, the 64-bit format spans in excess of 10^{-300} to 10^{+300} . This wide dynamic range, in conjunction with highly accurate rounding algorithms, renders the iSBX 332 module ideal for scientific problems and other applications requiring high speed, accuracy, and range. □

MULTIPROCESSING SYSTEM MIXES
8- AND 16-BIT MICROCOMPUTERS

April 1980

Multiprocessing System Mixes
8- and 16-Bit MicrocomputersJoseph P. Barthmaier
Computer Design, February 1980

MULTIPROCESSING SYSTEM MIXES 8- AND 16-BIT MICROCOMPUTERS

Combining different single-board computers on a single bus and assigning to each the tasks most suited enable a cost-effective multiprocessing system configuration with improved throughput and reliability

Joseph P. Barthmaier

Intel Corporation, Hillsboro, Oregon

Two or more single-board computers can share a common system bus to provide improved performance, reliability, and cost-effectiveness in medium to large scale applications. Interfacing multiple computers across a system bus affords a dual-bus architecture in which global system traffic is isolated from local traffic on the board buses. This allows a straightforward design of modular multiprocessing systems that combines different computer boards, and allocates to each that portion of the overall system function to which it is best suited.

In a typical design, 8- and 16-bit single-board computers (SBCs) communicate across a system bus to service an application that requires both realtime data acquisition and extensive signal processing. Partitioning system tasks and assigning each to the appropriate SBC optimizes performance without adding components. Dual-port memory provides a convenient way to synchronize

processes on different SBCs. Because most system functions are isolated on one SBC, reliability and throughput are increased, and implementation is facilitated.

Single-Board Computer Concept

In earlier SBC design, the fundamental goal was to provide a board containing all the resources required for a large variety of microprocessing applications. A typical processor board supplies an 8080A processor, 4k bytes of random access memory (RAM), sockets for up to 8k bytes of erasable programmable read only memory/read only memory (EPROM/ROM), a serial input/output (I/O) interface, 48 parallel I/O lines, three timer/counters, and eight levels of priority interrupt. With this hardware configuration, many small applications can be served with no need for additional memory or digital logic.

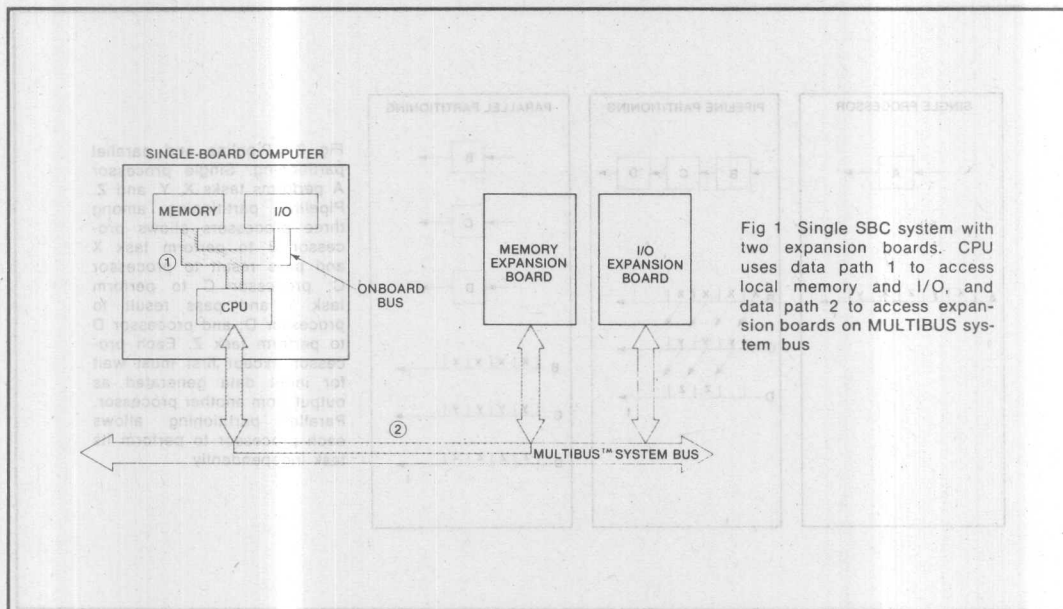


Fig 1 Single SBC system with two expansion boards. CPU uses data path 1 to access local memory and I/O, and data path 2 to access expansion boards on MULTIBUS system bus

Because larger applications require additional resources, an external bus structure was defined. The MULTIBUS™ system bus was designed for communication between SBCs and system expansion boards. Address, data, and handshake lines were defined for memory and I/O transfers between SBCs and expansion boards. There are bus expansion boards for system expansion in areas of RAM and ROM storage, serial and parallel I/O, analog I/O, and peripheral controllers.

In Fig 1, two buses interconnect a system with an SBC and two expansion boards. An onboard bus accesses local resources, and the system bus accesses global resources. A key advantage of this structure is that an SBC may not require the system bus for a large portion of its memory or I/O transactions. In many applications, less than 10% of the time is taken by system bus accesses. The large amount of potential system bus capacity makes this architecture a natural candidate for multiprocessing applications. As additional SBCs are included in the system, the incremental amount of system bus bandwidth required is usually small.

Motivations for Multiprocessing

Certain system applications benefit from using more than a single SBC. Motivations for constructing multiprocessing systems with SBCs include:

Resource sharing. In a multiprocessing system designed around the resource sharing concept, two or more processor boards share a common resource, such as a high

speed mathematics board or a peripheral controller. These boards perform independent functions with no relationship to one another except for the shared resource. Low cost is the obvious motivation for using a resource sharing multiprocessing configuration. If two processor boards share the same diskette controller, for example, overall system costs are considerably reduced.

Enhanced system throughput and performance. In many applications, significant improvements in performance may be achieved by using more than one processor in the system. Two ways of allocating or partitioning system functions among multiple processors, such as pipeline and parallel partitioning, are shown in Fig 2. In pipeline partitioning, system functions (tasks) are divided among several processors, so that data flow through the system is primarily serial. Each processor performs its portion of system functions, and then calls upon another processor to perform another set. An example of pipeline partitioning is when one processor performs data acquisition and buffering, while a second uses the data to perform digital signal processing.

Parallel partitioning allocates system functions among several processors in such a way that each processor performs a separate system task in parallel. An example is a system where one processor performs an industrial process control loop, while another monitors and controls a varying parameter, such as temperature.

Few systems may be characterized as totally parallel or pipeline partitioned, but designating systems in this manner can often be helpful during the system design phase, particularly when interprocessor communication software is being designed.

Modularly configured systems. A primary design goal, particularly in systems that are produced in low volume,

MULTIBUS is a registered trademark of Intel Corp.

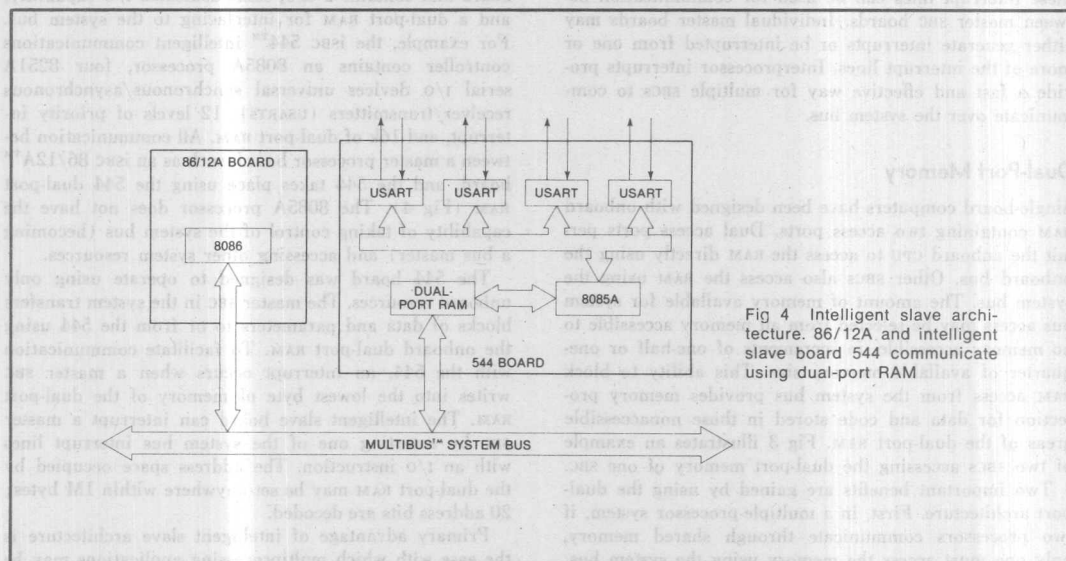


Fig 4 Intelligent slave architecture. 86/12A and intelligent slave board 544 communicate using dual-port RAM.

The 86/12A Board

The iSBC 86/12A single-board computer¹ has many of the architectural features of 8-bit boards (serial and parallel I/O, multiple interrupt levels, and timer/counters) but includes a 5-MHz 8086 microprocessor and larger amounts of RAM and EPROM/ROM storage. The 16-bit 8086 permits byte and word transfers, hardware multiply/divide, 1M-byte addressability, extensive string manipulation instructions, and many other features. The 86/12A contains 32k bytes of dual port RAM and sockets for up to 16k bytes of EPROM/ROM, doubling the memory available on previous boards. If more RAM or EPROM/ROM storage is required, memory expansion modules permit doubling RAM and/or EPROM/ROM storage to 64k bytes of RAM and 32k bytes of EPROM/ROM.

Memory expansion modules are small printed circuit boards that attach to the 86/12A board using sockets and nylon bolts. Use of the expansion modules is advantageous from a price/performance point of view. Price of either of the memory expansion modules is significantly less than that of an equivalent separate memory expansion board with its own system bus interface and support circuitry. Memory expansion modules also offer higher performance since it is not necessary to use the system bus for memory transactions. All transactions take place using the onboard bus with no additional wait states or bus contention.

8- and 16-bit MULTIBUS Compatibility

The 8086 microprocessor performs 8- or 16-bit transfers to or from memory or I/O devices. When a byte (8-bit) transfer is requested from an even address, data are pre-

sented to the microprocessor on its low order data lines, D0 through D7. When a byte transfer is requested from an odd address, data transfer must occur on the high order data lines, D8 through D15. When a 16-bit (word) transfer is requested, data are transferred on all 16 data lines, D0 through D15. When an 8-bit microprocessor (8080A or 8085A) is used, however, all byte transfers must take place on data lines D0 through D7, the only lines available.

To maintain compatibility between boards with 8-bit and 16-bit processors, a system bus transfer protocol has been developed where all byte transfers, regardless of whether from an odd or even address, take place on the low order system bus data lines, DAT0/ to DAT7/; word transfers, however, use all 16 data lines, DAT0/ to DATF/. To accomplish these byte transfers, an 8-bit buffer is used on 16-bit master and slave boards for transferring data from the high order data lines on the board to the low order data lines of the system bus. An additional signal line, byte high enable, (BHEN/) indicates whether a word transfer is taking place on the high order and low order data lines or whether a byte transfer is taking place only on the low order data lines. Fig 5 illustrates 8- and 16-bit transfers and the use of the additional buffer for transferring the signal to or from the high order data byte.

Multiprocessing System Example

A data acquisition and signal processing system design demonstrates the capabilities of a multiprocessing system, where improved performance is mandatory. General application of the system is power spectrum analysis of vibration and acoustic signals. Application areas for such

these interrupt lines can be used for communication between master SBC boards. Individual master boards may either generate interrupts or be interrupted from one or more of the interrupt lines. Interprocessor interrupts provide a fast and effective way for multiple SBCs to communicate over the system bus.

Dual-Port Memory

Single-board computers have been designed with onboard RAM containing two access ports. Dual access ports permit the onboard CPU to access the RAM directly using the onboard bus. Other SBCs also access the RAM using the system bus. The amount of memory available for system bus access may be selected from all memory accessible to no memory accessible, in increments of one-half or one-quarter of available memory size. This ability to block RAM access from the system bus provides memory protection for data and code stored in those nonaccessible areas of the dual-port RAM. Fig 3 illustrates an example of two SBCs accessing the dual-port memory of one SBC.

Two important benefits are gained by using the dual-port architecture. First, in a multiple-processor system, if two processors communicate through shared memory, only one must access the memory using the system bus, and the amount of system bus traffic may be significantly reduced. Second, in a multiprocessor configuration where limited RAM storage is required, a separate memory board is not needed. Such small systems have all the required system bus-accessible memory on one or more of the SBCs.

Intelligent Slave Architecture

To distribute intelligence in larger systems, the intelligent slave concept was developed. An intelligent slave is a

board that contains a CPU, some dedicated I/O capability, and a dual-port RAM for interfacing to the system bus. For example, the iSBC 544™ intelligent communications controller contains an 8085A processor, four 8251A serial I/O devices universal synchronous/asynchronous receiver/transmitters (USARTs), 12 levels of priority interrupt, and 16k of dual-port RAM. All communication between a master processor board, such as an iSBC 86/12A™ board, and the 544 takes place using the 544 dual-port RAM (Fig 4). The 8085A processor does not have the capability of taking control of the system bus (becoming a bus master) and accessing other system resources.

The 544 board was designed to operate using only onboard resources. The master SBC in the system transfers blocks of data and parameters to or from the 544 using the onboard dual-port RAM. To facilitate communication with the 544, an interrupt occurs when a master SBC writes into the lowest byte of memory of the dual-port RAM. The intelligent slave board can interrupt a master SBC by asserting one of the system bus interrupt lines with an I/O instruction. The address space occupied by the dual-port RAM may be set anywhere within 1M bytes; 20 address bits are decoded.

Primary advantage of intelligent slave architecture is the ease with which multiprocessing applications may be implemented. The intelligent slave may be sent a buffer of data and commands with an interrupt occurring, via a write to the lowest byte of memory, as a start command. The master SBC may continue operation with other functions to be notified, via an interrupt or a status byte in dual-port RAM, when the slave has completed a task. Since the intelligent slave may not access system resources via the system bus, no interference with the master SBC can occur.

iSBC and the combination of iSBC and a numerical suffix are registered trademarks of Intel Corp.

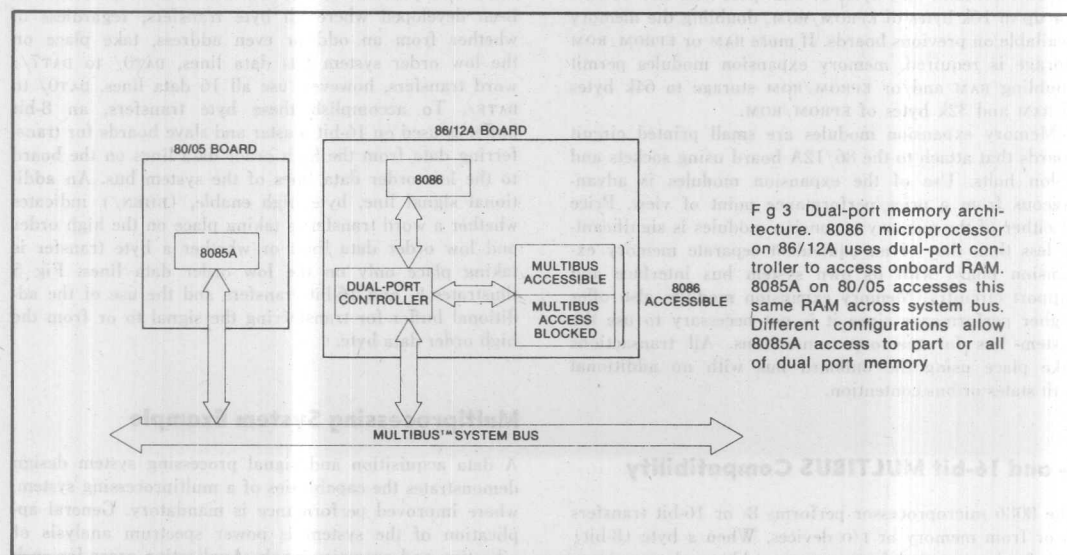


Fig 3 Dual-port memory architecture. 8086 microprocessor on 86/12A uses dual-port controller to access onboard RAM. 8085A on 80/05 accesses this same RAM across system bus. Different configurations allow 8085A access to part or all of dual port memory

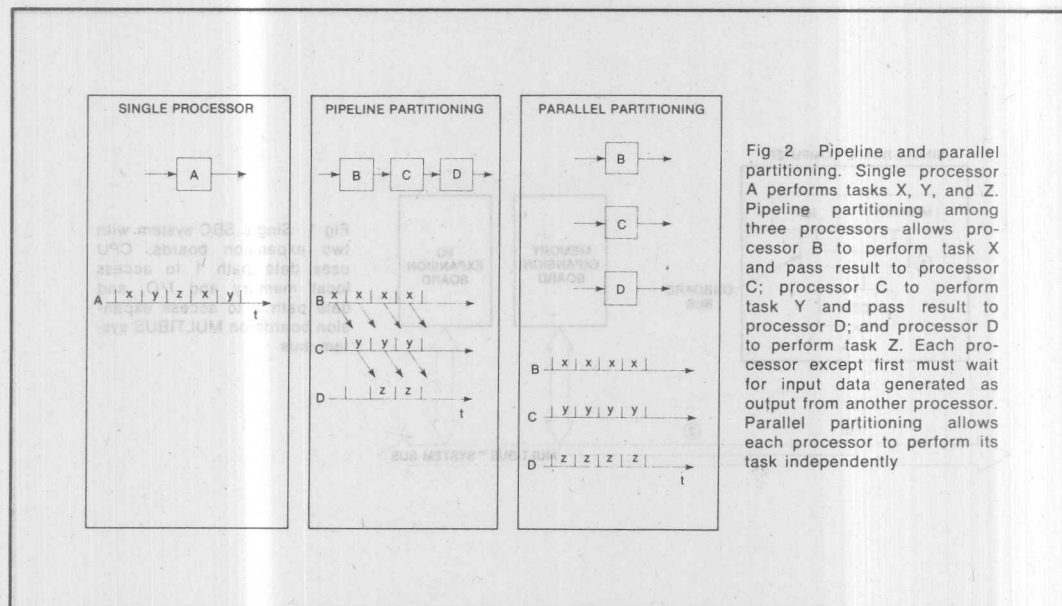


Fig 2 Pipeline and parallel partitioning. Single processor A performs tasks X, Y, and Z. Pipeline partitioning among three processors allows processor B to perform task X and pass result to processor C; processor C to perform task Y and pass result to processor D; and processor D to perform task Z. Each processor except first must wait for input data generated as output from another processor. Parallel partitioning allows each processor to perform its task independently

is often flexibility of system configuration. Using modularly configured systems, independent hardware and software modules are designed and implemented with individual processors or intelligent slave boards. When a particular configuration is required, the system designer selects the necessary hardware and software modules and combines them with interprocessor communication software. Shortened system development time, simple debugging, and a convenient upgrade path for system expansion are the benefits of such a technique.

High reliability. Multiprocessing may be used to isolate system tasks on individual processors in applications where a high degree of reliability is a requirement. If a processor fails, the remainder of the system continues to operate. Redundant designs, where a second processor may be dynamically assigned to perform the functions of a disabled processor, are a possibility.

Multiprocessing With Single-Board Computers

The MULTIBUS architectural design facilitates multiprocessing because multiple bus masters are accommodated, bus masters generate and acknowledge bus interrupts, and dual-port memory and intelligent slave architecture can be implemented.

Multiple Bus Masters

A bus master is a dynamic board that takes control of the bus by asserting address and control lines. Only one bus master may control the bus at any given moment. Examples of bus masters include single-board computers

and direct memory access (DMA) controllers. A bus slave is a passive element on the bus that does not assert address and control lines. Examples of bus slaves include memory or I/O expansion boards and intelligent slaves.

Several control lines exist on the bus so that potential bus masters can exchange control. These control lines, plus logic on the master boards, implement a priority scheme in which the highest priority master requesting the bus obtains control. There are two priority resolution schemes for exchange of the bus, serial and parallel. Using serial priority resolution, there may be up to three bus masters in the system; the parallel technique allows up to 16. A bus master is always given the opportunity to complete a bus transaction before being preempted by a higher priority master. In addition, bus masters may retain control of the bus by "locking" the bus. The bus lock feature is required when a master must have exclusive control of the bus for such functions as testing and setting software semaphores and completing operations involving I/O devices.

Since SBCs have extensive onboard resources, system bus transactions are not required for all I/O and memory accesses. Depending on the application and system design, multiple bus master systems with a small number of transactions can be configured. The system design goal is to use onboard resources whenever possible. Frequently executed or time critical code should be stored in onboard memory to minimize system bus accesses and to avoid delays while contending for the bus.

Interprocessor Interrupts

Eight interrupt lines exist on the system bus. In addition to interrupts from I/O slave boards or DMA controllers,

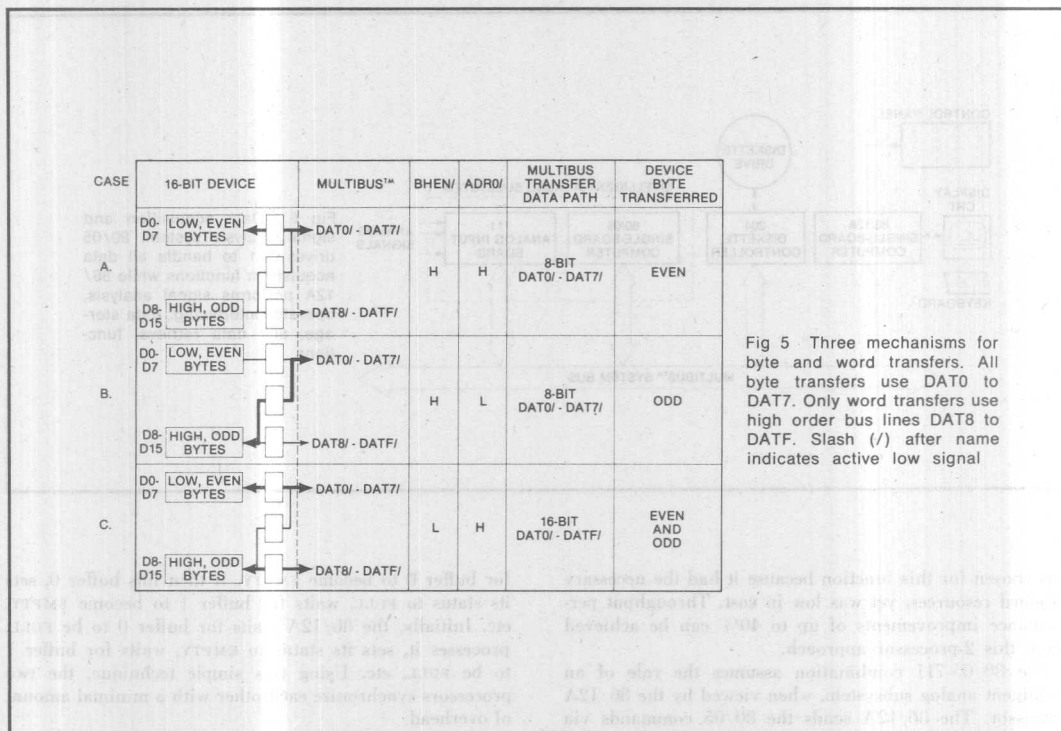


Fig 5 Three mechanisms for byte and word transfers. All byte transfers use DAT0 to DAT7. Only word transfers use high order bus lines DAT8 to DATF. Slash (/) after name indicates active low signal

systems include vibration analysis in mechanical structures such as electric motors, automobiles, aircraft, and buildings, as well as speech, sonar, and low frequency radar analysis.

Design objective is to monitor the condition of large electric motors. Power spectra of vibration signals from various points on the motor are calculated in order to detect bearing wear and to predict an impending motor failure. Calculated power spectra are compared with reference spectra, and, if thresholds in various regions of the spectra are exceeded, an operator alarm is activated. Information regarding the state of the motors and the reference spectra is stored on disc.

The system monitors 16 channels of analog input signals generated by pairs of accelerometers mounted on each of eight motors. Sampling and calculations for the two channels of a single motor are performed simultaneously; then the next motor in sequence is monitored.

Fast Fourier transform (FFT) of a buffer of samples from an analog to digital converter (ADC) performs power spectrum calculations. Real and imaginary parts of the FFT results are squared and summed to form a power spectrum that is compared to the reference power spectrum in order to determine if the motor vibrations are within acceptable tolerances. A CRT displays calculated and reference power spectra. At periodic intervals, data are stored on disc for archiving the condition of each of the motors. If the motor spectra exceed the reference

spectra, the CRT display and a control panel indicator alert the operator.

System Hardware

As shown in Fig 6, the 711 analog input board, containing a 12-bit ADC, samples the 16 analog signals from the motors. The 80/05 processor board drives the 711 analog board and handles all system data acquisition activities. The 80/05 contains an 8085A CPU, 512 bytes of RAM, up to 4k bytes of EPROM/ROM, a timer/counter, parallel and serial I/O lines, and four levels of priority interrupt. The 86/12A board is the main system processor. The 8086-based board performs all the signal processing functions, displays the spectra on the CRT, drives the system control panel, and transfers motor condition data onto disc using the 204 single-density diskette controller.

Increased system performance is the design motivation for using two processor boards. The 86/12A board, with its 5 MHz 8086 CPU, 16-bit multiply/divide capability, 64k bytes of dual-port RAM, and 32k bytes of EPROM/ROM, is used for the mathematically intensive power spectrum calculations.

The 80/05 processor board is used to offload data acquisition activities from the main processor. It assumes all the overhead of handling the 711 analog board. Sampling is performed at 250-μs intervals using the onboard timer; data from the two channels are scaled, demultiplexed, and stored in a buffer. The 8-bit processor board

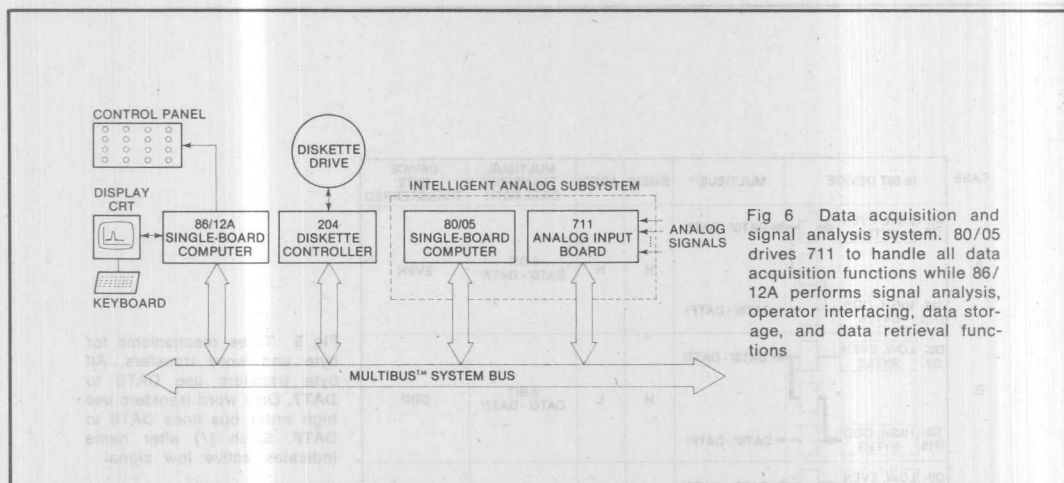


Fig 6 Data acquisition and signal analysis system. 80/05 drives 711 to handle all data acquisition functions while 86/12A performs signal analysis, operator interfacing, data storage, and data retrieval functions

was chosen for this function because it had the necessary onboard resources, yet was low in cost. Throughput performance improvements of up to 40% can be achieved using this 2-processor approach.

The 80/05-711 combination assumes the role of an intelligent analog subsystem, when viewed by the 86/12A processor. The 86/12A sends the 80/05 commands via a parameter block, and the 80/05 collects the data samples in buffers. When a buffer is complete, the 80/05 signals the 86/12A using the parameter block. Thus, the 80/05 acts as an intelligent DMA controller for the 711 board.

System Software

Due to the large RAM requirements of the system, the iSBC 300™ RAM expansion module is used to increase RAM capacity to 64k bytes. Memory has been configured to make 16k bytes of memory accessible to the system bus, with the remaining 48k bytes reserved for use by the onboard 8086 and not accessible to the system bus. The amount of 86/12A dual-port RAM that is system bus accessible may be configured in 16k increments from zero (no memory accessible) to 64k (all memory accessible). The parameter block used for interprocessor communication and a pair of buffers used for storage of the analog samples are stored in the memory accessible to the 80/05. Memory not accessible to the 80/05 contains the data and buffers used for the calculated averaged power spectra, reference spectra, CRT displays, and disc data. The 16 bytes of the parameter block contain all information required for communication between the two SBCs in the system, including buffer addresses, status, size, sample rate, and start and end channel.

Fig 7 is a flow diagram illustrating how buffer status bytes are used to synchronize the filling and processing of the data buffers. Each buffer may be in one of two states, FULL or EMPTY. Initially, both buffers are EMPTY. At initialization, the 80/05 fills buffer 0, sets its status to FULL, fills buffer 1, sets its status to FULL, and waits

for buffer 0 to become EMPTY. It then fills buffer 0, sets its status to FULL, waits for buffer 1 to become EMPTY, etc. Initially, the 86/12A waits for buffer 0 to be FULL, processes it, sets its status to EMPTY, waits for buffer 1 to be FULL, etc. Using this simple technique, the two processors synchronize each other with a minimal amount of overhead.

The parameter block approach is used to provide a simple means for interfacing the two SBCs. At system initialization, the 80/05 board needs only to know the base address of the parameter block. Once this is known, all other information required for the 80/05 to function properly is available. The end application and even the specific type of SBC that calls upon the 80/05 for data samples remain irrelevant to the 80/05. Driver software for the 80/05 is therefore highly modular and may be used in a variety of applications and configurations with no changes required.

A key capability of this system design is that the 86/12A board does not use the system bus to access data samples, thus minimizing execution time for the highly iterative FFT computation. The 80/05 processor takes the samples from the analog board and stores them directly into the 86/12A dual-port memory. Therefore, except for occasional disc transfers by the 86/12A, the 80/05 is the only processor using the system bus. This increases system throughput and eliminates contention for the system bus.

Signal Processing Software

The algorithm used for the FFT in this application is known as "time decomposition with input bit reversal."² Using this algorithm, an in-place FFT has been programmed for an input frame size of 128 complex points. Sixteen-bit integer mathematics is used for all internal calculations of the FFT. The 86/12A board computes the 128-point complex FFT in 110 ms. Computation of the averaged power spectra is performed using a double pre-

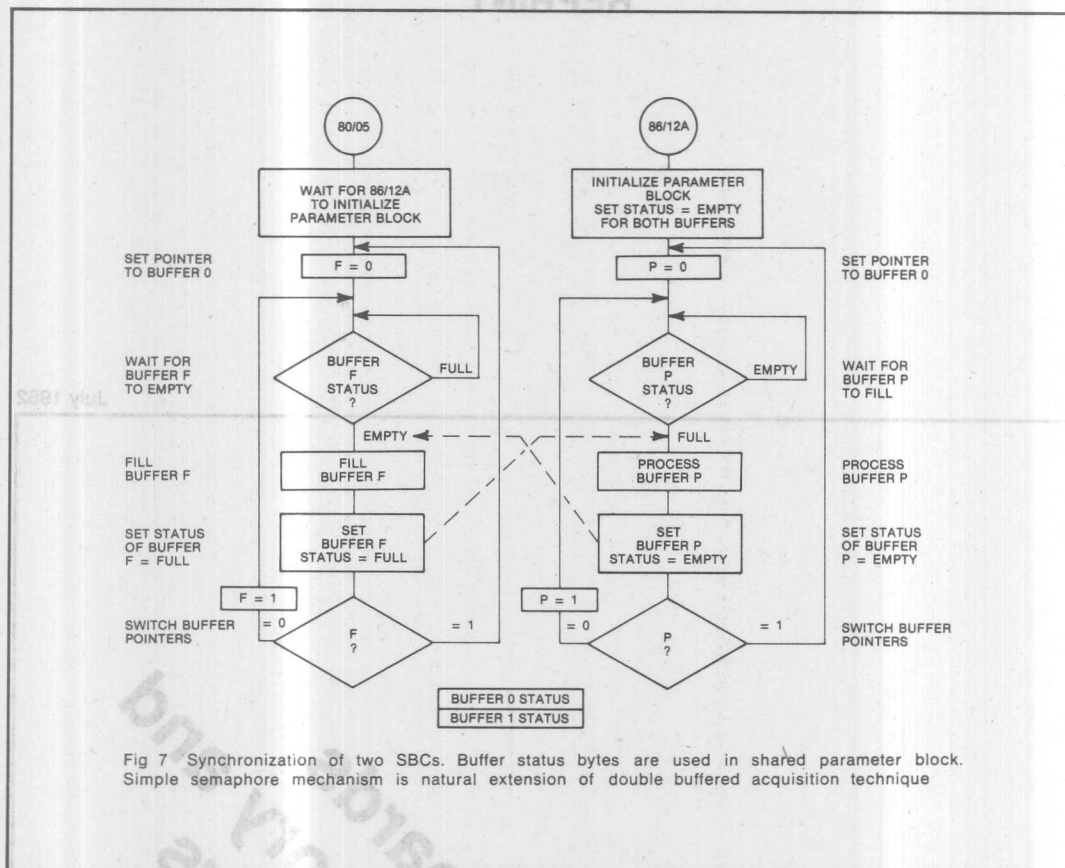


Fig 7 Synchronization of two SBCs. Buffer status bytes are used in shared parameter block. Simple semaphore mechanism is natural extension of double buffered acquisition technique

cision integer format. The 16-bit integer real and imaginary values which result from the FFT are squared and summed to obtain a 32-bit power spectrum. Thirty-two frames of data are processed and summed to form the averaged power spectrum.

Summary

Two reasons for the slow growth of multiprocessing have been the limited selection of SBCs and the relatively small application base. These conditions are changing rapidly due to the large number of SBCs now available. These boards contain dual-port RAM and newer 8- and 16-bit CPUs, and provide system designers with a comprehensive set of tools for tackling applications that require the power of multiprocessing. Thus, the SBC application base has grown significantly in recent years.

The system application combines a low cost 8-bit SBC and a high performance 16-bit SBC in a configuration designed for both data acquisition and signal analysis. The 8-bit SBC relieves the 16-bit SBC of all system data acquisition functions. Because the 16-bit board spends

full time processing data, system throughput can be increased by as much as 40%.

References

1. *iSBC 86/12A Hardware Reference Manual*, 98003074-01, Intel Corp, Santa Clara, Calif, 1979
2. S. D. Stearns, *Digital Signal Analysis*, Hayden Book Co, Rochelle Park, NJ, 1975



Joseph P. Barthmaier is applications engineering manager for the OEM Micro-computer Systems Operation at Intel. He is responsible for application notes and seminars covering the SBC product family and for future product planning. He holds BSEE and MSEE degrees from the University of Florida and Stanford University, respectively.

July 1982

Enhanced μ C Boards Strengthen Factory and Office Controllers

Gary Sawyer
Scott Tetrick
Don Peterson
Electronic Design

New single-board computers sport dramatic advances in density and speed. The upshot: much more capable factory and office controllers.

Enhanced μ C boards strengthen factory and office controllers

Designers of control and computer systems for the factory or office should be aware of the updated performance capabilities of the latest breed of single-board microcomputers. Faster microprocessors with wider word lengths are joining with denser, faster static memories to effect even finer control of time-critical operations. Yet the new crop maintains bus and software compatibility with earlier designs, needs fewer power supplies, and remarkably, charges no penalty for the extra work. In some cases, costs will dip.

Using such boards as the iSBC 86/XX family, designers can trade in 8 bits for 16 (see "A Bridge from 8- to 16-bit Processing"), 5-MHz operation for 8, older memory sockets for standard 28-pin JEDEC EPROM sockets. In addition, plug-on modules offer low-cost expansion. That adds up to double the EPROM capacity with no redesign and to quadruple the RAM capacity.

Moreover, the iSBX 86/XX boards support the full IEEE-796 Multibus specification—including 16-Mbyte addressing features (send and receive) and the use of the lock feature (Fig. 1). Four additional address lines on the Multibus can be used to address memories larger than 1 Mbyte. The lock function increases system speed—by preventing arbitration delays—and serves in semaphore signaling applications (for more particulars, see "Meet the Family").

In a multiprocessing system, individual processors must have a method of synchronization and mutual exclusion. A semaphore is the most common software method for providing these functions. Figure 2 shows an example of the semaphore concept and the need for dual locking. Assuming as in the

figure that the system consists of two processors and a disk-controller board, only one processor can access the mass-storage controller at any given time. A global memory byte indicates whether the disk-controller board is in use—a 1 indicates not busy; a 0, busy.

When a request is made for a disk file, the requesting processor must check the global byte to determine whether the controller is busy. Although the process appears simple, there could be pitfalls (Fig. 3a). The hardware must allow the processors to read the semaphore bit without the intervening write cycle, to prevent deadlocking.

Earlier boards needed global memory, not dual porting, to support semaphores easily. They relied on Multibus arbitration to prevent deadlock, but that required an additional memory. Moreover, the boards required much more hardware and software to ensure proper operation. With the introduction of the IEEE-796 Multibus specification, a lock line added to the system bus allows a much simpler implementation of semaphores in a dual-port memory (Fig. 3b).

The on-board processor can prevent access to the dual port, and system accesses can lock the local processor out of the dual port. Both functions must be implemented to ensure the integrity of the semaphore byte. Figure 4 shows the arbitration scheme for a dual port.

Other board enhancements include Schmitt-trigger inputs on all critical Multibus command lines, which improves noise immunity. Pull-up resistors on all input lines allow for easy testing. The new board layouts minimize the length of the Multibus signal lines, reducing the line's susceptibility to crosstalk pickup. These improvements result in a mean time between failures (MTBF) that is 50% greater than that of comparable first-generation boards.

Most measures of a computer board's processing power are summed up in one specification, the clock

Gary Sawyer, Product Manager
Scott Tetrick, Product Engineer
Don Peterson, Marketing Engineer
Intel Corp.
5200 N.E. Elam Young Pkwy.
Hillsboro, Ore. 97123

Enhanced single-board control

rate. With an 8-MHz 8086-2 microprocessor on board, processing speeds up 60% over 5-MHz operation.

Because of increased processing speed, on-board circuitry must be upgraded to handle higher data and instruction rates. Thus, some control functions are implemented with programmable logic arrays (PLAs). These high-speed devices not only replace dedicated chips, but also allow logic operations to be performed at clock rates exceeding 10 MHz.

Stepping up the speed

To accommodate the demand for greater data handling, RAM access time is 750 ns, compared with 1000 ns in earlier single-board computers. An additional benefit of faster access time is the capacity for dual-port access. With this feature, dual-port RAMs can be accessed from the Multibus in 500 ns when locked and 800 ns when unlocked. High-speed, dual-port access permits interprocessor communications to occur at a much faster rate.

Since each iSBC board contains a different RAM storage capacity, speed performance is linked closely to a board's memory capacity. As Fig. 5 shows, numbers on the vertical scale indicate increasing performance for a given storage capacity.

The timing/storage performance graphs of Fig. 5 have been determined by making certain assumptions about system operation. First, it is assumed that the system has no Multibus contention for the

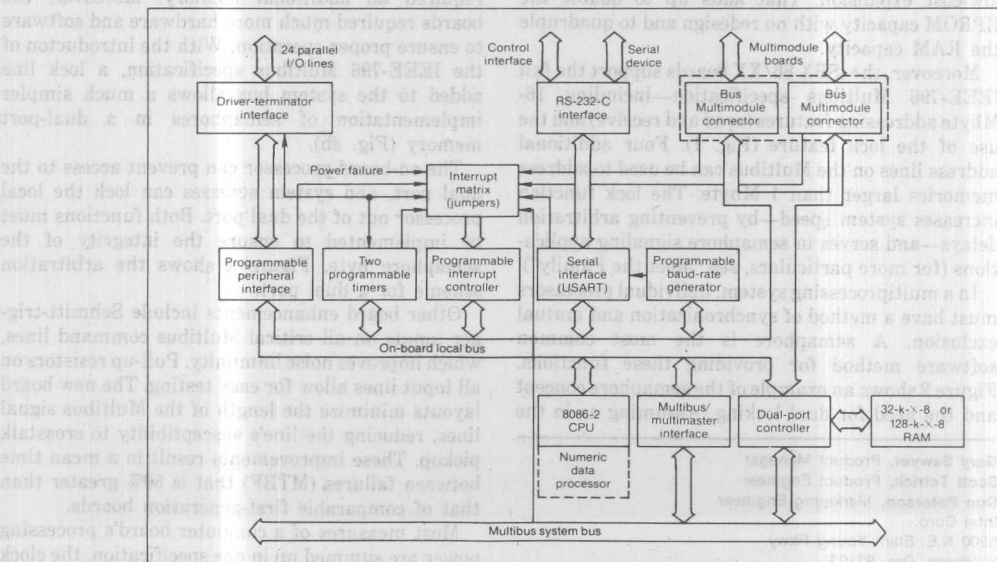
global memory as is the case in a single-master system. Also, all memory is accessed constantly, independent of on- or off-board optimization. For example, because of delays encountered in acquiring the Multibus, data rates plunge (in the iSBC 86/05) if more processors are added to a system.

The expanded memory space—320 kbytes—can serve either for enhanced multiprocessing or for the storage of large amounts of application code. The new design prevents performance-reducing arbitration delays in multiple-processor system operation. Proper partitioning of the system memory allows code accesses from the on-board memory. Since code fetching operations occupy most of the system-bus bandwidth, execution from local memory greatly accelerates system throughput.

At home in factory or office

Two key application areas—and their diverse requirements—stand out for single-board computers. One area includes machine and process control, instrumentation, and specialized data acquisition. Here, high-performance single-board computers are required to have extensive EPROM capacity to provide sufficient program storage without relying on mass storage.

The other area covers communications systems, small-business computers, and word and data processing. Single-board computers in office-of-the-



1. Advances in digital LSI and VLSI have expanded the capabilities of single-board computers. The result is more speed and more memory, yet compatibility with earlier versions.

future applications are generally RAM-intensive, with programs downloaded from a disk. A small amount of EPROM serves to bootstrap the system RAM at power-up. Mass storage is used to hold the large data bases required, but the computer board must support an I/O terminal for the human interface. Typically, word-processing systems require a high-performance CPU, a large RAM capacity, an interface for mass storage, a CRT terminal, and a printer.

For RAM-intensive applications, a 128-kbyte RAM module fits on board. An on-board serial I/O port and 24 programmable I/O lines provide the interface with the CRT display and printer. With the inclusion of the iSBX bus on the board, low-cost I/O expansion is possible. Also, the iSBX 218 single- or double-density, single- or double-sided floppy-disk controller allows the entire system to fit on a single board. Moreover, even with those components on the board, there is room for a mathematics module. So the same basic system can serve as a high-performance small-business computer.

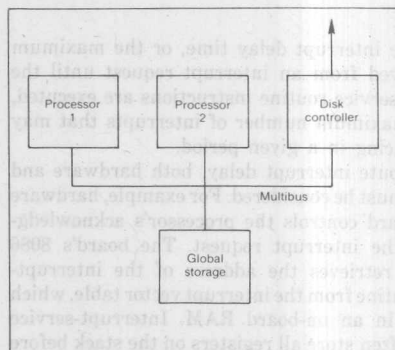
In high-speed communications applications, a high-end single-board controller outfitted with an iSBX module offers an interface capable of handling processing needs well into the future. Since the iSBX module communicates directly with the CPU over the iSBX bus, Multibus contention does not pose a problem. Another benefit of the bus-module approach: reduced size and power consumption.

Factory-control considerations

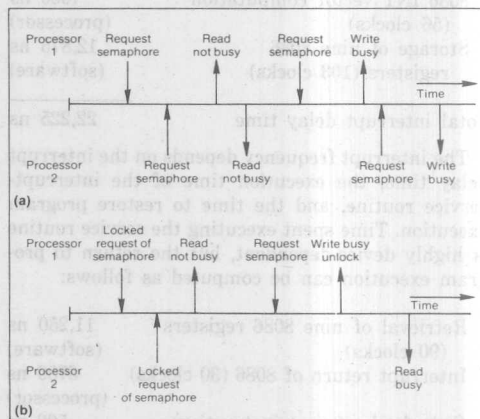
When the single-board controller is to be used to control various phases of a manufacturing process, two things must be considered. First, operations are time-critical. The time between samples in a manufacturing process defines the maximum interval during which defective products could be made. Often, the volume of production requires the use of multiple processors.

Second, factory operations do not occur at pre-determined intervals, but are event-driven. In addition, code operations are usually fixed, since the factory environment is quite predictable. Thus, code can be stored in PROM or EPROM, but that requires enough ROM storage capability on the controller board. Finally, specialized operations are common, the most often used being floating-point arithmetic for three-dimensional movement and analog input and output for manipulating machinery.

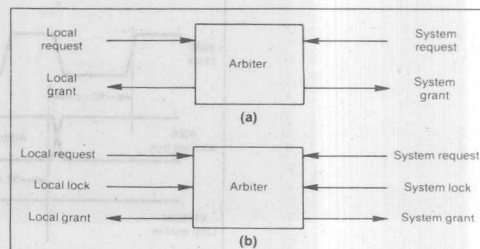
To equip a general-purpose board as a factory controller, its interrupt structure, PROM, and I/O access must all be high-speed, and the board must have a range of configurable options. After selecting a board, the designer should determine his maximum interrupt-speed requirement by measuring two



2. Semaphore signaling is a method of synchronizing the operation of processors in a multiprocessing system. In an earlier single-board system (shown here), global storage provides the control signal for indicating the status of the disk controller—busy or not busy.



3. In newer single-board computers, a locking feature eliminates the need for global storage in semaphore signaling. Without locking, a deadlock can occur (a) when two processors try to access a system resource. The dual-port memory prevents such a problem (b).



4. Adding dual-port memory locking to computer boards inhibits arbitration until the locked operation is completed (a). Earlier boards relied on global storage (b).

Enhanced single-board control

things: the interrupt delay time, or the maximum time allowed from an interrupt request until the interrupt service routine instructions are executed, and the maximum number of interrupts that may need servicing in a given period.

To compute interrupt delay, both hardware and software must be considered. For example, hardware on the board controls the processor's acknowledgment of the interrupt request. The board's 8086 processor retrieves the address of the interrupt-service routine from the interrupt vector table, which is stored in an on-board RAM. Interrupt-service routines often store all registers on the stack before the servicing of an interrupting device begins. The total delay time is computed as follows:

INT to 8259A to INTR at 8086	350 ns
INTA service time (16 8086 clocks)	2000 ns
	(hardware)
8086 INT vector computation	7000 ns
(56 clocks)	(processor)
Storage of nine 8086 registers (103 clocks)	12,875 ns
	(software)

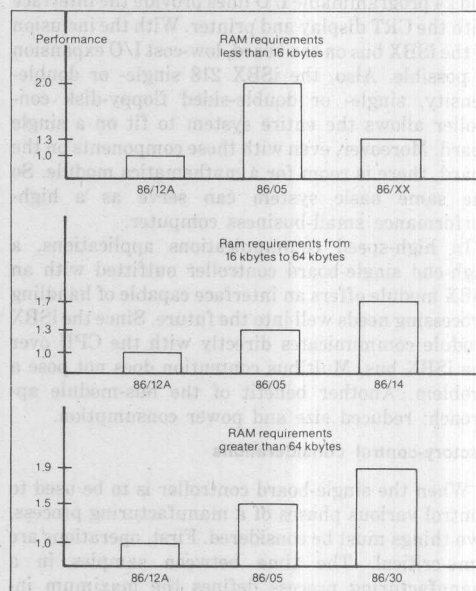
Total interrupt delay time 22,225 ns

The interrupt frequency depends on the interrupt delay time, the execution time of the interrupt-service routine, and the time to restore program execution. Time spent executing the service routine is highly device-dependent, but the return to program execution can be computed as follows:

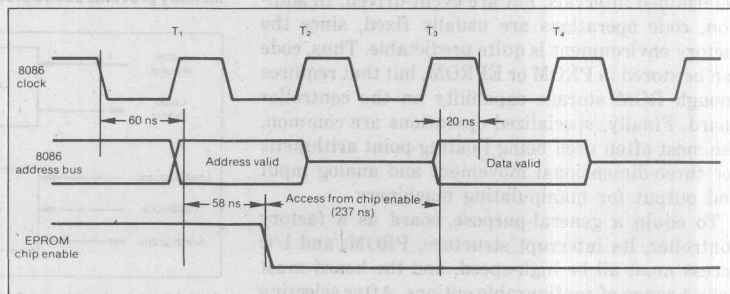
Retrieval of nine 8086 registers (90 clocks)	11,250 ns
	(software)
Interrupt return of 8086 (30 clocks)	3750 ns
	(processor)
Code fetch of next instruction (four clocks)	500 ns
	(hardware)
Total return time	15,500 ns

Interrupt frequencies for the iSBC 86/14—assuming several I/O service routines—are 24,600 per second for sending a character to a CRT, 24,600/s for receiving a character from the CRT, and 22,000/s for clock interrupts.

The access times of current EPROMs range from 200 to 1000 ns. For slower devices, wait states (an integer number of processor clock times) are added to the access time. At an 8-MHz processing frequency, a 200-ns EPROM operates with zero wait states. The most critical time, access time from a chip



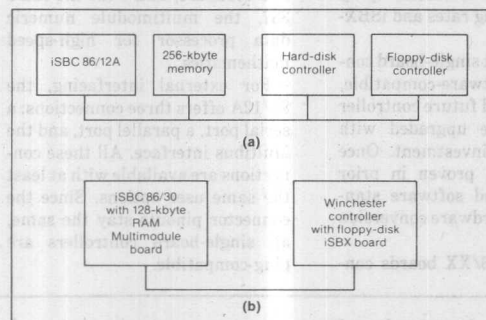
5. The random-access-memory performance of single-board computers is related to the amount of storage a board provides. The latest boards are designed to work most efficiently with storage capacities of 16 to 256 kbytes.



6. Factory controllers execute most of the system code from EPROMs. With a high-speed EPROM, valid data appears on the Multibus about 300 ns after valid addresses are applied.

enable, is 200 ns. The timing relationships for accessing an EPROM are shown in Fig. 6.

From a system standpoint, all peripheral components can be accessed in about 300 ns after a command. At a clock rate of 5 MHz, that forces one wait state; at 8 MHz, two wait states. A variety of devices interfaces with the iSBX bus, permitting custom configurations of a single-board controller for factory applications. Two analog modules, one for input and one for output, can monitor or control plant equipment. For expansion of digital inputs or



7. Considerable hardware can be saved when a single-board computer replaces earlier versions (a) in an office controller. One advantage of the new boards (b) is their large on-board storage capacity, which allows code to be stored locally, increasing system speed and throughput.

outputs, a parallel I/O module can be added.

Whereas interrupt response time is a critical parameter in a factory environment, other considerations become critical in an office environment. One of those is throughput, the number of tasks a worker can perform. As a result, office controllers must provide the user with fast response time. In addition, since office controllers usually perform a wide variety of tasks, execution is mainly from RAM, not ROM. Moreover, because most tasks are dynamic, large mass-storage devices are needed. Programs are loaded from mass storage into the processor's memory and executed. In addition, most programs are in a high-level language (for word processing and other office tasks), which requires considerable hardware and software support.

As with factory controllers, configurable systems are important in the office environment, since special applications will require additional hardware. Word processors, for example, are generally linked via a serial communications line to a central processor, or mass-storage device, and the link must be a high-speed one to minimize response time.

To meet office system requirements, computer boards should have the following features: an operating system having a human interface (keyboard), a program loader and configurable peripheral support for mass storage, communications controllers and high-level languages; a maximum-size RAM with minimum access-time devices for program support;

A bridge from 8- to 16-bit processing

Until recently, most dedicated controller applications had to be handled with an 8-bit microcomputer because of price and performance considerations. But with performance and software needs on the rise, designers are faced with moving up to 16-bit devices to obtain the necessary computing power. When the move up happens, however, hardware and software compatibility considerations often get the short end of the stick.

The iSBC 88/25 could be termed the missing link between 8-bit buses and 16-bit processing power. Using the 8088 microprocessor, the board interfaces with the outside world on an 8-bit bus, but internally it operates as a 16-bit system. Since software compilers such as the PL/M-86 often generate 16-bit data-access instructions, a designer using a 16-bit 8086-based board must have a 16-bit memory expansion board or pay a burdensome software penalty. What's more, an 8086-based board must access instructions in 16-bit words, eliminating the possibility of using 8-bit boards for instruction storage.

The iSBC 88/25 automatically breaks all 16-bit words into 8-bit bytes on the Multibus. Therefore, independently of software, the hardware converts all accesses to 8-bit bytes transparently to the user. This gives the board the dual advantage of being fully compatible with all 8-bit hardware devices and 16-bit software. To retain compatibility with Intel standards, the board is designed to interface with the Multibus system bus, the iSBX bus, memory modules, and the iSBC 337 math module and numeric data processor. The board also contains JEDEC-compatible 28-pin sockets to accommodate EPROM and static byte-wide RAM.

The 8088 microprocessor is being used in the second generation of personal computers, such as the one from IBM. Several other personal-computer and small-business systems will also use the 8088. With its software compatibility with the 8086 processor and widespread application in personal and small-business systems, the software repertoire of the 8088 will exceed that of any microprocessor.

and hardware support for both mass storage and communications.

Office controllers call for lots of RAM, and a board with a capacity of up to 256 kbytes is suitable. A dynamic RAM controller handles the automatic refreshing and generates the proper timing for the

row and column addresses. PLA devices send commands to the controller to initiate the RAM access.

The differences between early and updated office controllers are shown in Fig. 7. Additional memory expansion on the Multibus is necessary to meet system requirements. The floppy-disk controller

Meet the family

Progress in semiconductor density and speed translates into more functions and greater throughput in the latest breed of single-board computers. Another, more subtle, benefit accrues: board-level controllers can be made to fit within a wide range of performance and cost slots, resulting in greater freedom of design choice.

Thus whereas one board, the iSBC 86/05, provides high-speed, low-cost compatibility with earlier units, two other boards, the iSBC 86/14 and iSBC 86/30, drop into the high-performance end of system integration. Both con-

troller boards are memory-intensive: the former is available with 32 kbytes of RAM, expandable to 64 kbytes; the latter starts with 128 kbytes of RAM and can be expanded to 256 kbytes. Large storage capacity is backed up by 8-MHz processing rates and ISBX-bus support.

Since all 16-bit single-board controllers are software-compatible, both present and future controller systems can be upgraded with minimal extra investment. Once established and proven in prior designs, bus and software standards reduce hardware conversion costs.

Since iSBC 86/XX boards con-

stitute a family, hardware and software are interchangeable throughout. For example, the earlier iSBC 86/12A is compatible with the ICE 86, the in-circuit emulator for debugging hardware and software, and with the iSBC 337, the multimodule numeric data processor for high-speed mathematics.

For external interfacing, the 86/12A offers three connections: a serial port, a parallel port, and the Multibus interface. All these connections are available with at least the same user options. Since the connector pinouts stay the same, all single-board controllers are plug-compatible.

board alone must be implemented with a full size Multibus board. Using newer boards not only reduces system size, but also improves performance. The

large on-board memory stores enough program information to allow operation that is 100% faster than first-generation boards.□

The iRMX 86 operating system standard, together with its application software, dictates that all single-board controllers have the same memory-access capability and I/O resources. To ensure compatibility between the earlier and the newer boards, the new iSBC 86/14 is designed with the same memory capabilities as the iSBC 86/12A for on- and off-board access to memory. In addition, the iSBC 86/30, with its 128 kbytes of RAM, further extends the system's capabilities. I/O controllers are contained on iSBC 86/XX boards in the same default configuration as on iSBC 86/12A boards. Thus, the software—like

the hardware—is plug-compatible, virtually eliminating the need for additional software expense when upgrading a system.

If a system's RAM requirements are under 16 kbytes, an iSBC 86/05 supported by a high-speed 8-kbyte RAM Multimodule board (iSBC 302), offers better performance than either an 86/12A or any newer board. For factory applications, the 86/05 offers twice the EPROM capacity of the 86/14 or 30 board. But as RAM requirements grow past 16 kbytes, the 86/05 becomes less desirable because the additional memory required must be accessed from the Multibus. Data accesses from the

bus occur at a much slower rate than those from the on-board memory.

For midrange RAM storage applications—16 to 64 kbytes—there is the iSBC 86/14 and a 32-kbyte RAM Multimodule board (iSBC 300A). The 86/30 runs as fast as the 86/14, but for RAM requirements exceeding 64 kbytes, the 86/30 continues to access its internal memory, providing faster operation. Moreover, an expanded memory space—a total of 320 kbytes consisting of 256 kbytes of RAM and 64 kbytes of EPROM—make the 86/30 the performance leader in large system applications.

High Speed Math Boards

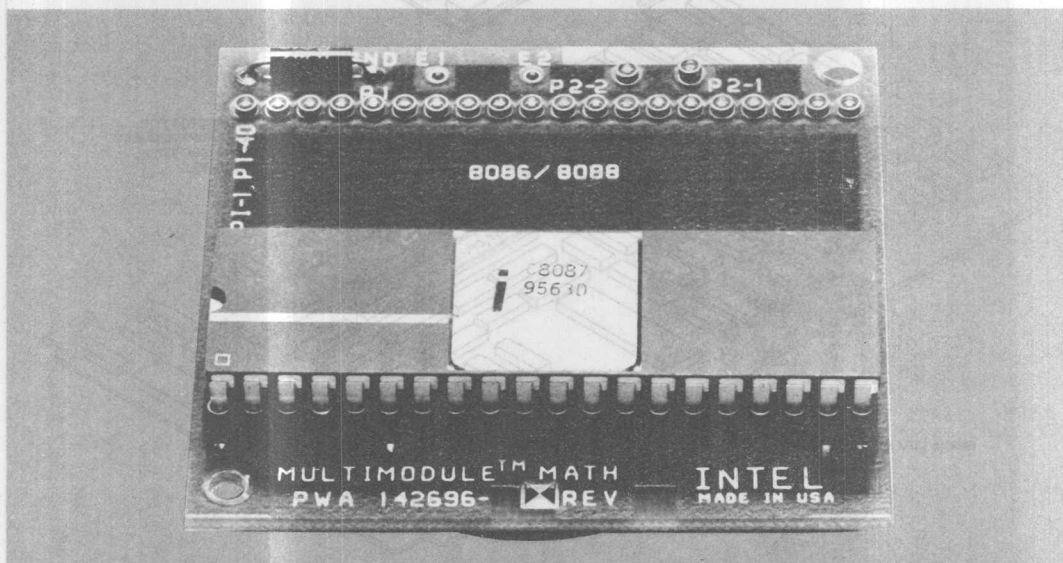
4



iSBC™ 337 MULTIMODULE™ NUMERIC DATA PROCESSOR

- High speed fixed and floating point functions for iSBC 86, 88 and iAPX 86, 88 systems
- MULTIMODULE option containing 8087 Numeric Data Processor
- Supports seven data types including single and double precision integer and floating point
- Implements proposed IEEE Floating Point Standard for high accuracy
- Extends host CPU instruction set with arithmetic, logarithmic, transcendental and trigonometric instructions
- 50 x performance improvements in Whetstone benchmarks over iAPX 86/10 performance
- Software support through ASM-86/88 Assembly Language and High Level Languages

The Intel iSBC 337 MULTIMODULE Numeric Data Processor offers high performance numerics support for iSBC 86 and iSBC 88 Single Board Computer users, for applications including simulation, instrument automation, graphics, signal processing and business systems. The coprocessor interface between the 8087 and the host CPU provides a simple means of extending the instruction set with over 60 additional numeric instructions supporting six additional data types. The data formats conform to the proposed IEEE Floating Point Standard insuring highly accurate results. The MULTIMODULE implementation allows the iSBC 337 module to be used on all iSBC 86 and iSBC 88 Microcomputers and can be added as an option to custom iAPX 86 and iAPX 88 board designs.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Index, Intel, MULTIBUS, RMX, IRMX, UPI, ICE, iSBC, iSBX, MULTIMODULE, iAPX and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

intel

The iSBC 337 MULTIMODULE Numeric Data Processor provides arithmetic and logical instruction extensions to the 8086 and 8088 CPU's of the iAPX 86 and iAPX 88 families, to provide iAPX 86/20 and iAPX 88/20 Numeric Data Processors. The instruction set consists of arithmetic, transcendental, logical, trigonometric and exponential instructions which can all operate on seven different data types. The data types are 16, 32, and 64 bit integer, 32 and 64 bit floating point, 18 digit packed BCD and 80 bit temporary.

Coprocessor Interface

The coprocessor interface between the host CPU (8086 and 8088) and the iSBC 337 processor provides easy to use and high performance math processing. Installation of the iSBC 337 processor is simply a matter of removing the host CPU from its

socket, installing the iSBC 337 processor into the host's CPU socket, and reinstalling the host CPU chip into the socket provided for it on the iSBC 337 processor (see Figure 1). All synchronization and timing signals are provided via the coprocessor interface with the host CPU. The two processors also share a common address/data bus. (See Figure 2.) The 8087 Numeric Data Processor (NDP) component is capable of recognizing and executing 8087 numeric instructions as they are fetched by the host CPU. This interface allows concurrent processing by the host CPU and the 8087. It also allows 8087 and host CPU instructions to be intermixed in any fashion to provide the maximum overlapped operation and the highest aggregate performance.

High Performance and Accuracy

The 80-bit wide internal registers and data paths contribute significantly to high performance and

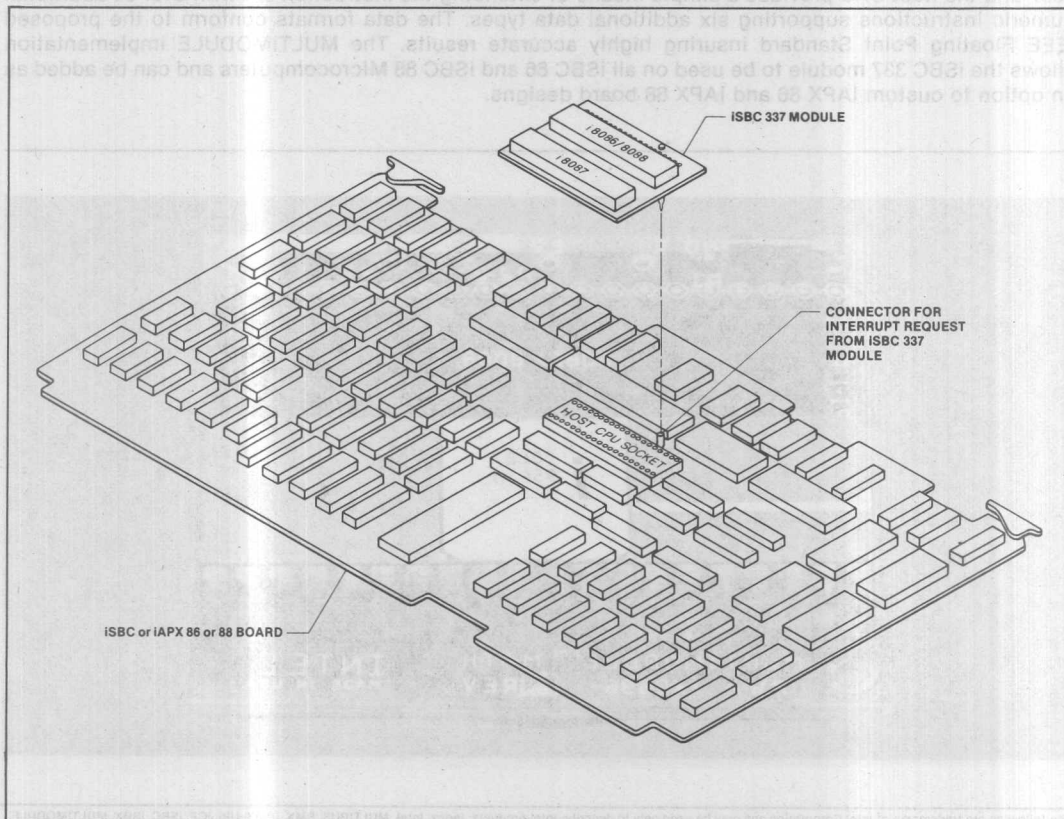


Figure 1. iSBC 337 Module Installation

minimizes the execution time difference between single and double precision floating point formats. This 80-bit architecture, in conjunction with the use of the proposed IEEE Floating Point Standard provides very high resolution and accuracy. This precision is complemented by extensive exception detection and handling. Six different types of exceptions can be reported and handled by the 8087. The user also has control over internal precision, infinity control and rounding control.

SYSTEM CONFIGURATION

As a coprocessor to an 8086 or 8088, the 8087 is wired in parallel with the CPU as shown in Figure 2. The CPU's status and queue status lines enable the NDP to monitor and decode instructions in synchronization with the CPU and without any

CPU overhead. Once started, the 8087 can process in parallel with and independent of the host CPU. For resynchronization, the NDP's BUSY signal informs the CPU that the NDP is executing an instruction and the CPU WAIT instruction tests this signal to insure that the NDP is ready to execute subsequent instructions. The NDP can interrupt the CPU when it detects an error or exception. The interrupt request line is routed to the CPU through an 8259A Programmable Interrupt Controller. This interrupt request signal is brought down from the ISBC 337 module to the ISBC 86, 88 Single Board Computer through a single pin connector (see Figure 1). The signal is then routed to the interrupt matrix for jumper connection to the 8259A Interrupt Controller. Other iAPX 86 and 88 designs may use a similar arrangement, or by masking off the 8086's "READ" pin from the ISBC 337 socket, provisions are made to allow the now vacated pin of the host's CPU socket to be used to bring down

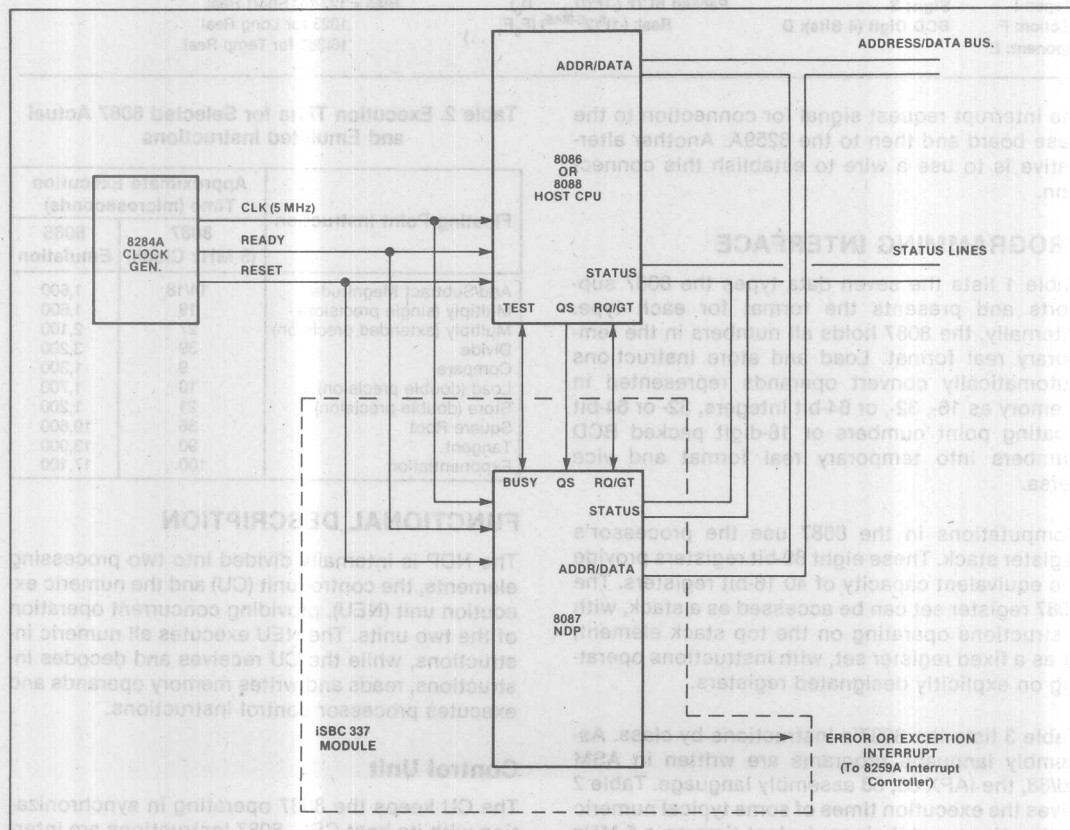


Figure 2. ISBC 337 System Configuration

Table 1. 8087 Datatypes

Data Formats	Range	Precision	Most Significant Byte											
			7	07	07	07	07	07	07	07	07	07	07	0
Word Integer	10 ⁴	16 Bits	<div>I₁₅ I₀</div> Two's Complement											
Short Integer	10 ⁹	32 Bits	<div>I₃₁ I₀</div> Two's Complement											
Long Integer	10 ¹⁹	64 Bits	<div>I₆₃ I₀</div> Two's Complement											
Packed BCD	10 ¹⁸	18 Digits	<div>S — D₁₇ D₁₆ D₁ D₀</div>											
Short Real	10 ^{±38}	24 Bits	<div>S E₇ E₀ F₁ F₂₃</div> F ₀ Implicit											
Long Real	10 ^{±308}	53 Bits	<div>S E₁₀ E₀ F₁ F₅₂</div> F ₀ Implicit											
Temporary Real	10 ^{±4932}	64 Bits	<div>S E₁₄ E₀ F₀ F₆₃</div>											

Note:

Integer: I Sign: S Packed BCD: $(-1)^S(D_{17} \dots D_0)$ Bias = 127 for Short Real
 Fraction: F BCD Digit (4 Bits): D Real: $(-1)^S(2^{E-BIAS})(F_0 F_1 \dots)$ 1023 for Long Real
 Exponent: E 161383 for Temp Real

the interrupt request signal for connection to the base board and then to the 8259A. Another alternative is to use a wire to establish this connection.

PROGRAMMING INTERFACE

Table 1 lists the seven data types the 8087 supports and presents the format for each type. Internally, the 8087 holds all numbers in the temporary real format. Load and store instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point numbers or 18-digit packed BCD numbers into temporary real format and vice versa.

Computations in the 8087 use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 40 16-bit registers. The 8087 register set can be accessed as a stack, with instructions operating on the top stack element, or as a fixed register set, with instructions operating on explicitly designated registers.

Table 3 lists the 8087's instructions by class. Assembly language programs are written in ASM 86/88, the iAPX 86, 88 assembly language. Table 2 gives the execution times of some typical numeric instructions and their equivalent time on a 5 MHz 8086.

Table 2. Execution Time for Selected 8087 Actual and Emulated Instructions

Floating Point Instruction	Approximate Execution Time (microseconds)	
	8087 (5 MHz Clock)	8086 Emulation
Add/Subtract Magnitude	14/18	1,600
Multiply (single precision)	19	1,600
Multiply (extended precision)	27	2,100
Divide	39	3,200
Compare	9	1,300
Load (double precision)	10	1,700
Store (double precision)	21	1,200
Square Root	36	19,600
Tangent	90	13,000
Exponentiation	100	17,100

FUNCTIONAL DESCRIPTION

The NDP is internally divided into two processing elements, the control unit (CU) and the numeric execution unit (NEU), providing concurrent operation of the two units. The NEU executes all numeric instructions, while the CU receives and decodes instructions, reads and writes memory operands and executes processor control instructions.

Control Unit

The CU keeps the 8087 operating in synchronization with its host CPU. 8087 instructions are intermixed with CPU instructions in a single instruc-

Table 3. 8087 Instruction Set

Data Transfer Instructions		Arithmetic Instructions		Processor Control Instructions	
Real Transfers		Addition			
FLD	Load real	FADD	Add real	FINIT/FNINIT	Initialize processor
FST	Store real	FADDP	Add real and pop	FDISI/FNDISI	Disable interrupts
FSTP	Store real and pop	FIADD	Integer add	FENI/FNENI	Enable interrupts
FXCH	Exchange registers			FLDCW	Load control word
Integer Transfers		Subtraction		FSTCW/FNSTCW	Store control word
FILD	Integer load	FSUB	Subtract real	FSTSW/FNSTSW	Store status word
FIST	Integer store	FSUBP	Subtract real and pop	FCLEX/FNCLEX	Clear exceptions
FISTP	Integer store and pop	FISUB	Integer subtract	FSTENV/FNSTENV	Store environment
Packed Decimal Transfers		FSUBR	Subtract real reversed	FLDENV	Load environment
FBLD	Packed decimal (BCD) load	FSUBRP	Subtract real reversed and pop	FSAVE/FNSAVE	Save state
FBSTP	Packed decimal (BCD) store and pop	FISUBR	Integer subtract reversed	FRSTOR	Restore state
Comparison Instructions		Multiplication		FINCSTP	Increment stack pointer
FCOM	Compare real	FMUL	Multiply real	FDECSTP	Decrement stack pointer
FCOMP	Compare real and pop	FMULP	Multiply real and pop	FFREE	Free register
FCOMPP	Compare real and pop twice	FIMUL	Integer multiply	FNOP	No operation
FICOM	Integer compare	Division		FWAIT	CPU wait
FICOMP	Integer compare and pop	FDIV	Divide real		
FTST	Test	FDIVP	Divide real and pop		
FXAM	Examine	FIDIV	Integer divide		
Transcendental Instructions		FDIVR	Divide real reversed		
FPTAN	Partial tangent	FDIVRP	Divide real reversed and pop		
FPATAN	Partial arctangent	FIDIVR	Integer divide reversed		
F2XM1	$2^X - 1$	Other Operations			
FYL2X	$Y \cdot \log_2 X$	FSQRT	Square root		
FYL2XP1	$Y \cdot \log_2(X + 1)$	FSCALE	Scale		
		FPREM	Partial remainder		
		FRNDINT	Round to integer		
		FTRACT	Extract exponent and significand		
		FABS	Absolute value		
		FCHS	Change sign		

tion stream. The CPU fetches all instructions from memory; by monitoring the status signals emitted by the CPU, the NDP control unit determines when an 8086 instruction is being fetched. The CU taps the bus in parallel with the CPU and obtains that portion of the data stream.

After decoding the instruction, the host executes all opcodes but ESCAPE (ESC), while the 8087 executes only the ESCAPE class instructions. (The first five bits of all ESCAPE instructions are identical). The CPU does provide addressing for ESC instructions, however.

An 8087 instruction either will not reference memory, will require loading one or more operands from memory into the 8087, or will require storing one or more operands from the 8087 into memory. In the first case a non-memory reference escape is used to start 8087 operation. In the last two cases, the CU makes use of a "dummy read" cycle initiated by the CPU, in which the CPU calculates the operand address and initiates a bus cycle, but does not capture the data. Instead, the CPU captures and saves the address which the CPU places on the bus. If the

instruction is a load, the CU additionally captures the data word when it becomes available on the local data bus. If data required is longer than one word, the CU immediately obtains the bus from the CPU using the request/grant protocol and reads the rest of the information in consecutive bus cycles. In a store operation, the CU captures and saves the store address as in a load, and ignores the data word that follows in the "dummy read" cycle. When the 8087 is ready to perform the store, the CU obtains the bus from the CPU and writes the operand starting at the specified address.

Numeric Execution Unit

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 80 bits wide (64 fraction bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the 8087 BUSY signal. This signal is

used in conjunction with the CPU WAIT instruction to resynchronize both processors when the NEU has completed its current instruction.

Register Set

The 8087 register set is shown in Figure 3. Each of the eight data registers in the 8087's register stack is 80 bits wide and is divided into "fields" corresponding to the NDP's temporary real data type.

The register set may be addressed as a push down stack, through a top of stack pointer or any register may be addressed explicitly relative to the top of stack.

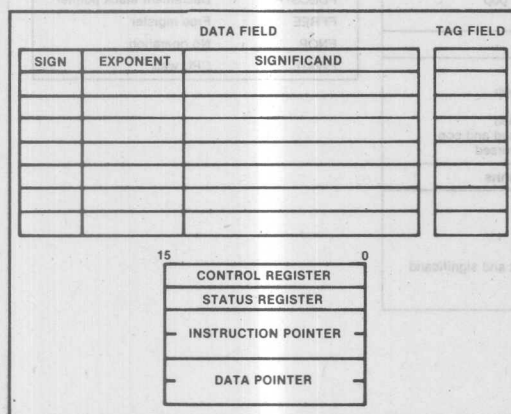


Figure 3. 8087 Register Set

Status Word

The status word shown in Figure 4 reflects the overall state of the 8087; it may be stored in memory and then inspected by CPU code. The status word is a 16-bit register divided into fields as shown in Figure 4. The busy bit (bit 15) indicates whether the NEU is executing an instruction (B = 1) or is idle (B = 0). Several instructions which store and manipulate the status word are executed exclusively by the CU, and these do not set the busy bit themselves.

The four numeric condition code bits (C_0 - C_3) are similar to the flags in a CPU: various instructions update these bits to reflect the outcome of NDP operations.

Bits 14-12 of the status word point to the 8087 register that is the current top-of-stack (TOP).

Bit 7 is the interrupt request bit. This bit is set if any unmasked exception bit is set and cleared otherwise.

Bits 5-0 are set to indicate that the NEU has detected an exception while executing an instruction.

Tag Word

The tag word marks the content of each register as shown in Figure 5. The principal function of the tag word is to optimize the NDP's performance. The tag word can be used, however, to interpret the contents of 8087 registers.

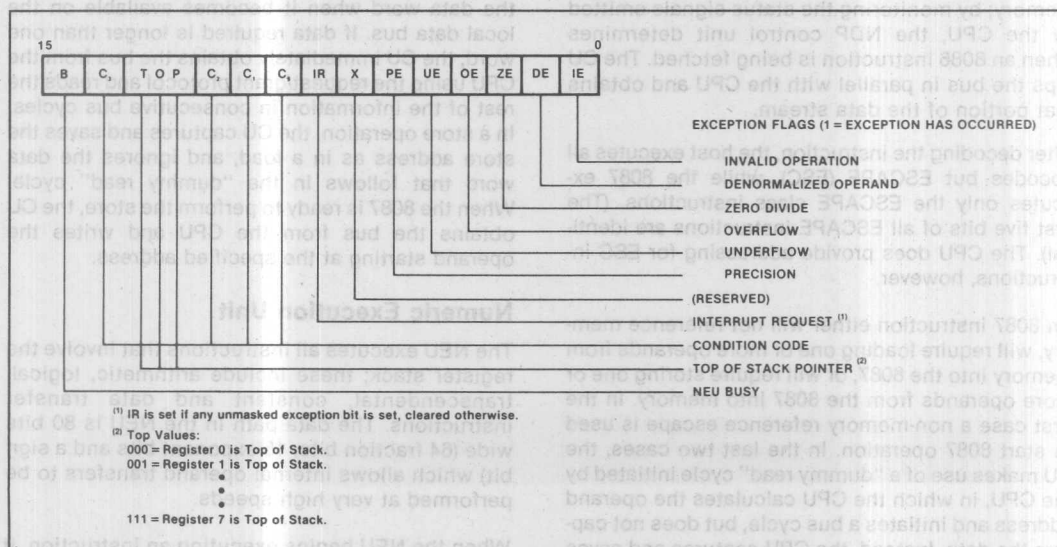


Figure 4. 8087 Status Word

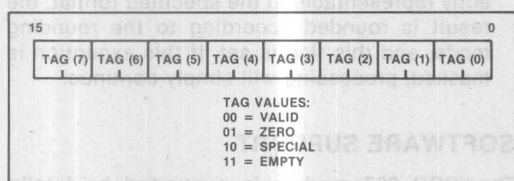


Figure 5. 8087 Tag Word

Instruction and Data Pointers

The instruction and data pointers (see Figure 6) are provided for user-written error handlers. Whenever the 8087 executes an NEU instruction, the CU saves the instruction address, the operand address (if present) and the instruction opcode. The 8087 can then store this data in memory.

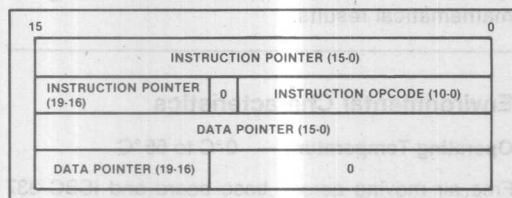


Figure 6. 8087 Instruction and Data Pointers

Control Word

The NDP provides several processing options which are selected by loading a word from memory into the control word. Figure 7 shows the format and encoding of the fields in the control word.

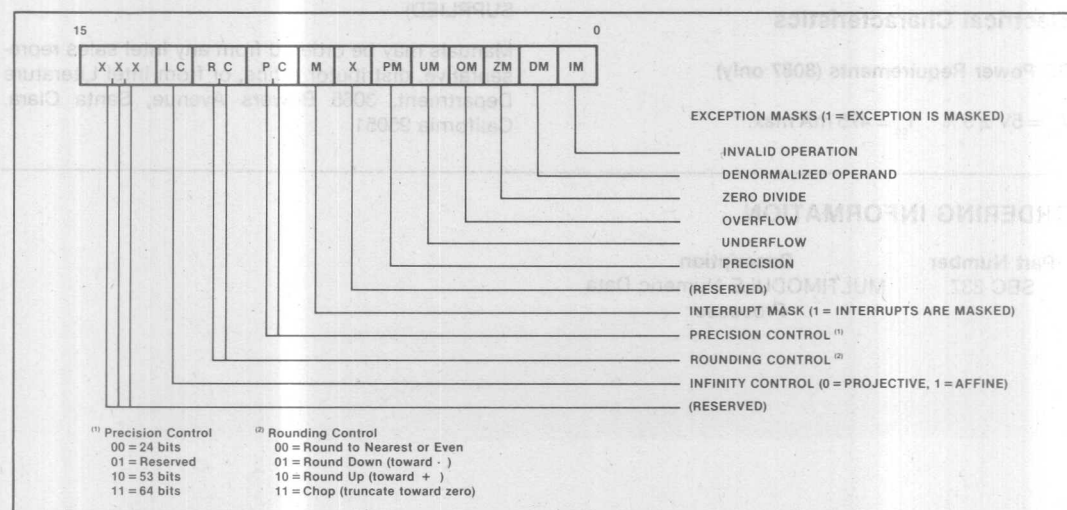


Figure 7. 8087 Control Word

Exception Handling

The 8087 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause an interrupt if unmasked and interrupts are enabled.

If interrupts are disabled the 8087 will simply suspend execution until the host clears the exception. If a specific exception class is masked and that exception occurs, however, the 8087 will post the exception in the status register and perform an on-chip default exception handling procedure, thereby allowing processing to continue. The exceptions that the 8087 detects are the following:

- 1. INVALID OPERATION:** Stack overflow, stack underflow, indeterminate form (0/0, —, etc.) or the use of a Non-Number (NAN) as an operand. An exponent value is reserved and any bit pattern with this value in the exponent field is termed a Non-Number and causes this exception. If this exception is masked, the 8087's default response is to generate a specific NAN called INDEFINITE, or to propagate already existing NANs as the calculation result.
- 2. OVERFLOW:** The result is too large in magnitude to fit the specified format. The 8087 will generate the code for infinity if this exception is masked.

3. ZERO DIVISOR: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 8087 will generate the code for infinity if this exception is masked.

4. UNDERFLOW: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 8087 will denormalize (shift right) the fraction until the exponent is in range. This process is called gradual underflow.

5. DENORMALIZED OPERAND: At least one of the operands or the result is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.

6. INEXACT RESULT: If the true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

SOFTWARE SUPPORT

The iSBC 337 module is supported by Intel's ASM-86/88 Assembly Language and PLM-86/88 Systems Implementation Language. In addition to the instructions provided in the languages to support the additional math functions, a software emulator is also available to allow the execution of iAPX 86/20 instructions without the need for the iSBC 337 module. This allows for the development of software in an environment without the iAPX 86/20 processor and then transporting the code to its final run time environment with no change in mathematical results.

SPECIFICATIONS

Physical Characteristics

Width — 5.33 cm (2.100")

Length — 5.08 cm (2.000")

Height — 1.82 cm (.718")
iSBC 337 board +
host board

Weight — 17.33 grams (.576 oz.)

Electrical Characteristics

DC Power Requirements (8087 only)

$V_{cc} = 5V \pm 5\%$ $I_{cc} = 475$ mA max.

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Free air moving across base board and iSBC 337 module.

Relative Humidity — Up to 90% R.H. without condensation.

Reference Manual

142887-001 — iSBC 337 MULTIMODULE Numeric Data Processor Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

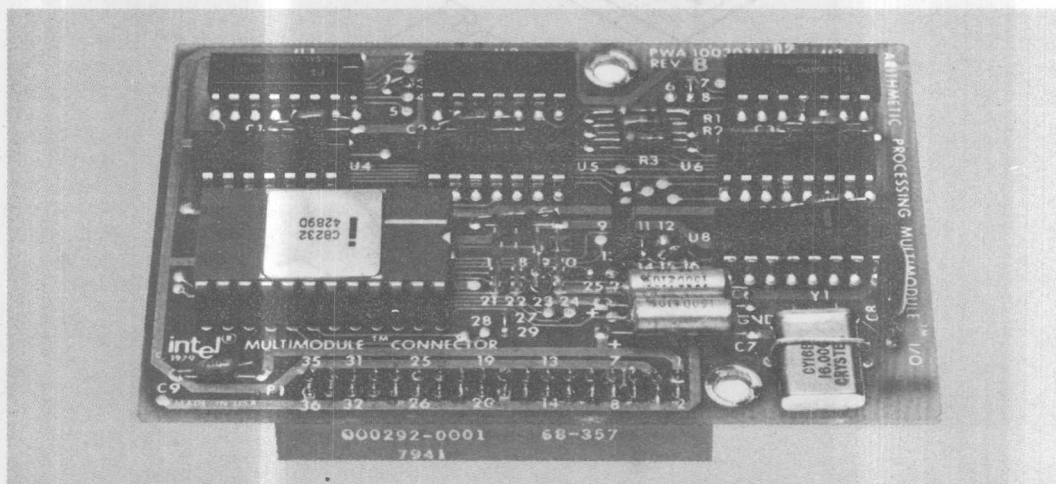
Part Number	Description
SBC 337	MULTIMODULE Numeric Data Processor



ISBX 332 FLOATING POINT MATH MULTIMODULE BOARD

- ISBX bus compatible high speed floating point math expansion
- 4 MHz operation
- Compatible with proposed IEEE format and existing Intel floating point standard
- Single (32-bit)/double (64-bit) precision arithmetic and data manipulation commands
- Performs functions independently and concurrently with the MULTIBUS host board
- Add, subtract, multiply and divide functions
- End-of-operation and error interrupts
- Software reset control
- Accessed as I/O port locations
- Low power requirements
- ISBX bus on-board expansion eliminates MULTIBUS system bus latency and increases system throughput

The Intel® ISBX 332 Floating Point Math MULTIMODULE Board is a member of Intel's new line of ISBX bus compatible MULTIMODULE products. The ISBX MULTIMODULE board plugs directly into any ISBX bus compatible host board offering incremental on-board expansion. The ISBX 332 module performs single (32-bit) and double (64-bit) precision floating point add, subtract, multiply, and divide functions compatible with the proposed IEEE floating point standard. The command operations run entirely independent of the host board permitting efficient concurrent processing. The ISBX board is closely coupled to the host board through the ISBX bus, and as such, offers maximum on-board performance and frees MULTIBUS system traffic for other system resources. In addition, incremental power dissipation is minimal requiring only 2.73 watts



FUNCTIONAL DESCRIPTION

The iSBX 332 module uses the Intel® 8232 Floating Point Processor (FPP) to accomplish high speed math operation. The system software may communicate with the iSBX 332 module across the iSBX bus using I/O read/write commands. All transfers, including operand, result, status, and command information, take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack and commands are issued to perform operations on the data stack. Results are then available to be retrieved from the stack. A status byte may be read to monitor execution completion and the nature of the result (zero, sign, or errors). In addition, control logic is included on the iSBX 332 module to facilitate single instruction software reset control.

Command Functions

The iSBX 332 module commands fall into three categories: single precision arithmetic, double precision arithmetic and data manipulation (see Table 1). There are four arithmetic operations that can be performed with single precision (32-bit) or double precision (64-bit) floating point numbers: add, subtract, multiply and divide. These operations require two operands. The 8232 assumes that these operands are located in the internal stack as Top of Stack (TOS) and Next on Stack (NOS). The result will always be returned to the previous NOS which becomes the new TOS. Results from an operation are of the same precision and format as the operands.

The results will be rounded to preserve the accuracy. In addition to the arithmetic operations, the 8232 implements eight data manipulating operations. These include changing the sign of a double or single precision operand located in TOS, exchanging single precision operands located at TOS and NOS, as well as copying and popping single or double precision operands. See also the sections on status register and operand formats.

The execution times of the commands are all data dependent. Table 2 shows one example of each command execution time.

Interrupt Requests

There are two interrupt lines from the FPP that may generate an interrupt request to the host: END (MINTR1) and ERINT (MINTR0). The END interrupt line is active upon command completion and the ERINT line is active when the current command execution results in an error condition. The error conditions are: attempt to divide by zero, exponent overflow and exponent underflow. Both the END and ERINT signals are cleared by a reset or status register read.

Installation

The iSBX 332 module plugs directly into the female iSBX connector on the host board. The module is then secured at one additional point with nylon hardware to insure the mechanical security of the assembly (see Figures 1 and 2).

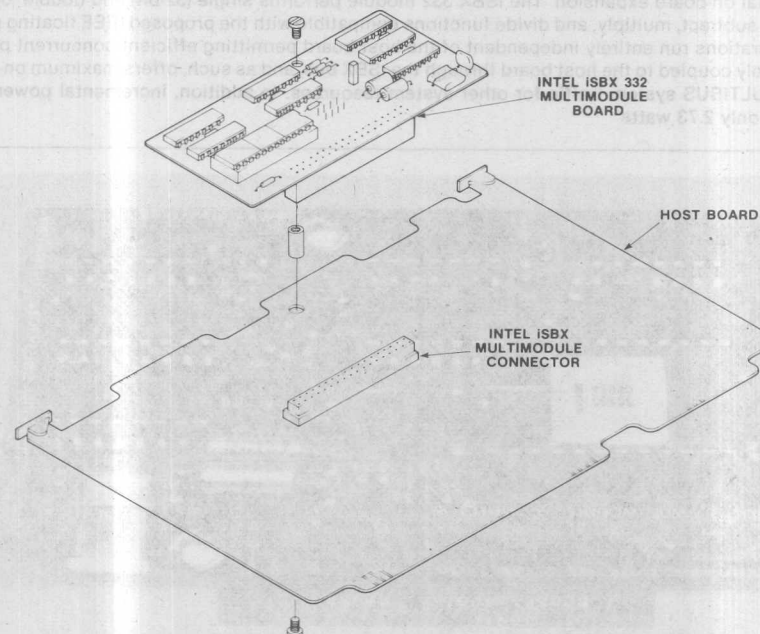


Figure 1. Installation of iSBX 332 Module on a Host Board

ISBX 332

Table 1. Command Summary

Command Bits	Mnemonic	Description
7 6 5 4 3 2 1 0		
X 0 0 0 0 0 0 1	SADD	Add TOS to NOS single precision and result to NOS. Pop stack.
X 0 0 0 0 0 1 0	SSUB	Subtract TOS from NOS single precision and result to NOS. Pop stack.
X 0 0 0 0 0 1 1	SMUL	Multiply NOS by TOS single precision and result to NOS. Pop stack.
X 0 0 0 0 1 0 0	SDIV	Divide NOS by TOS single precision and result to NOS. Pop stack.
X 0 0 0 0 1 0 1	CHSS	Change sign of TOS single precision operand.
X 0 0 0 0 1 1 0	PTOS	Push single precision operand on TOS to NOS.
X 0 0 0 0 1 1 1	POPS	Pop single precision operand from TOS. NOS becomes TOS.
X 0 0 0 1 0 0 0	XCHS	Exchange TOS with NOS single precision.
X 0 1 0 1 1 0 1	CHSD	Change sign of TOS double precision operand.
X 0 1 0 1 1 1 0	PTOD	Push double precision operand on TOS to NOS.
X 0 1 0 1 1 1 1	POPD	Pop double precision operand from TOS. NOS becomes TOS.
X 0 0 0 0 0 0 0	CLR	CLR status.
X 0 1 0 1 0 0 1	DADD	Add TOS to NOS double precision and result to NOS. Pop stack.
X 0 1 0 1 0 1 0	DSUB	Subtract TOS from NOS double precision and result to NOS. Pop stack.
X 0 1 0 1 0 1 1	DMUL	Multiply NOS by TOS double precision and result to NOS. Pop stack.
X 0 1 0 1 1 0 0	DDIV	Divide NOS by TOS double precision and result to NOS. Pop stack.

NOTE:

X= Don't care. Operation for bit combinations not listed above is undefined.

Table 2. Execution Times

Command	TOS	NOS	Result	Clock Periods	Time (μs)
SADD	3F800000	3F800000	40000000	58	14.5
SSUB	3F800000	3F800000	00000000	56	14.0
SMUL	40400000	3FC00000	40900000	198	49.5
SDIV	3F800000	40000000	3F000000	228	57.0
CHSS	3F800000	—	BF800000	10	2.5
PTOS	3F800000	—	—	16	4.0
POPS	3F800000	—	—	14	3.5
XCHS	3F800000	40000000	—	26	6.5
CHSD	3FF0000000000000	—	BFF0000000000000	24	6.0
PTOD	3FF0000000000000	—	—	40	10.0
POPD	3FF0000000000000	—	—	26	6.5
CLR	3FF0000000000000	—	—	4	1.0
DADD	3FF00000A0000000	8000000000000000	3FF00000A0000000	578	144.5
DSUB	3FF00000A0000000	8000000000000000	3FF00000A0000000	578	144.5
DMUL	BFF8000000000000	3FF8000000000000	C002000000000000	1748	437.0
DDIV	BFF8000000000000	3FF8000000000000	BFF0000000000000	4560	1140.0

NOTE:

TOS, NOS and result are in hexadecimal; clock period is in decimal.

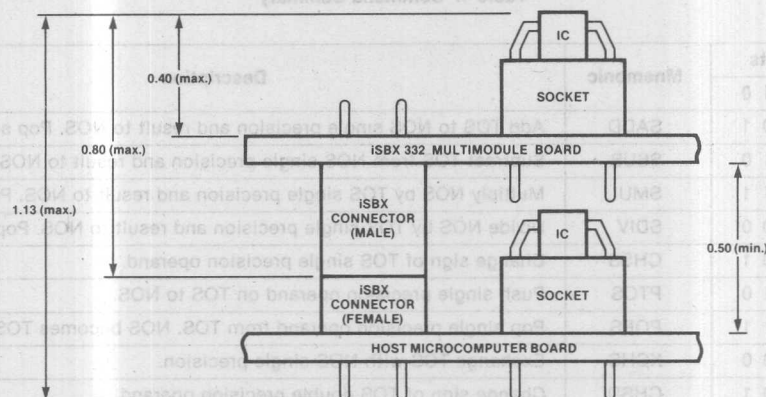


Figure 2. Mounting Clearances (Inches)

SPECIFICATIONS

Word Size

Data — 8 Bits

I/O Addressing

Function	Type of Operation	ISBX Connector Port Address
Data Transfer	Read or Write	X0, X2, X4, or X6
Command Transfer	Write	X1, X3, X5, or X7
Status Transfer	Read	X1, X3, X5, or X7
Reset	Write	X8 through XF

NOTE:

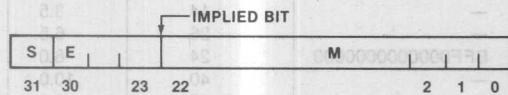
The port addresses are determined on the host ISBC microcomputer. Refer to the Hardware Reference Manual for your host ISBC microcomputer to determine the first digit (X) of the connector port address.

Arithmetic Functions

See Table 1

Floating Point Format

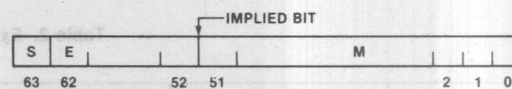
Single Precision Floating Point (32 Bits)



- Bit 31: S = Sign of the mantissa. 1 represents negative and 0 represents positive.
- Bits 23-30: E = These 8 bits represent a biased exponent. The bias is $2^7 - 1 = 127$.
- Bits 0-22: M = 23-bit mantissa. Together with the sign bit, the mantissa represents a signed fraction in sign-magnitude notation. There is an implied 1 beyond the most significant bit

(bit 22) of the mantissa. In other words, the mantissa is assumed to be a 24-bit normalized quantity and the most significant bit, which will always be 1 due to normalization, is implied. The FPP restores this implied bit internally before performing arithmetic, normalizes the result, and strips the implied bit before returning the results to the external data bus. The binary point is between the implied bit and bit 22 of the mantissa.

Double Precision Floating Point (64 Bits)



- Bit 63: S = Sign of the mantissa. 1 represents negative and 0 represents positive.
- Bits 52-62: E = Biased exponent. The bias is $2^{10} - 1 = 1023$.
- Bits 0-51: M = 51-bit mantissa. Together with the sign bit, the mantissa represents a signed fraction in sign-magnitude notation. There is an implied 1 beyond the most significant bit (bit 51) of the mantissa. In other words, the mantissa is assumed to be a 53-bit normalized quantity and the most significant bit, which will always be a 1 due to normalization, is implied. The FPP restores this implied bit internally before performing arithmetic, normalizes the result, and strips the implied bit before returning the result to the external data bus. The binary point is between the implied bit and bit 51 of the mantissa.

Status Byte

Contains the following information:

BUSY	SIGN S	ZERO Z	RESERVED	DIVIDE EXCEPTION D	EXPONENT UNDERFLOW U	EXPONENT OVERFLOW V	RESERVED
7	6	5	4	3	2	1	0

- Bit 0 Reserved.
- Bit 1 Exponent Overflow (V): When 1, this bit indicates that exponent overflow has occurred. Cleared to zero otherwise.
- Bit 2 Exponent Underflow (U): When 1, this bit indicates that exponent underflow has occurred. Cleared to zero otherwise.
- Bit 3 Divide Exception (D): When 1, this bit indicates that an attempt to divide by zero has been made. Cleared to zero otherwise.
- Bit 4 Reserved.
- Bit 5 Zero (Z): When 1, this bit indicates that the result returned to TOS after a command is all zeros. Cleared to zero otherwise.
- Bit 6 Sign (S): When 1, this bit indicates that the result returned to TOS is negative. Cleared to zero otherwise.
- Bit 7 Busy: When 1, this bit indicates the APU is in the process of executing a command. It will become zero after the command execution is complete. All other status bits should be considered to be undefined if this bit is set.

Access Time

Read — 1900 ns (max.)

Write — 1900 ns (max.)

NOTE:

Actual transfer speed is dependent upon the cycle time of the host microcomputer. The listed times assume no operation in progress. If an operation is executing when an access is attempted, the command execution time must be added to the above times for all accesses except status read.

Interrupts

Two interrupt requests may originate from the FPP indicating command completion (END) and error conditions (ERINT).

Interface

iSBX Bus — All signals TTL compatible

Physical Characteristics

Width — 6.35 cm (2.50 in.)

Length — 9.40 cm (3.70 in.)

Height* — 2.04 cm (0.80 in.) iSBX 332 Board

— 2.86 cm (1.13 in.) iSBX 332 Board + Host Board

Weight — 51 gm (1.79 oz)

*See Figure 2

Electrical Characteristics

DC Power Requirements

$V_{CC} = +5V \pm 5\%$

$I_{CC} = 365 \text{ mA max.}$

$V_{DD} = +12V \pm 5\%$

$I_{DD} = 75 \text{ mA max.}$

Environmental

Operating Temperature — 0°C to 55°C

Free moving air across the base board and iSBX board.

Reference Manual

9803204-01 — iSBX 332 Floating Point Math MULTIMODULE Board (NOT SUPPLIED)

Reference Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Ave., Santa Clara, California 95051.

ORDERING INFORMATION

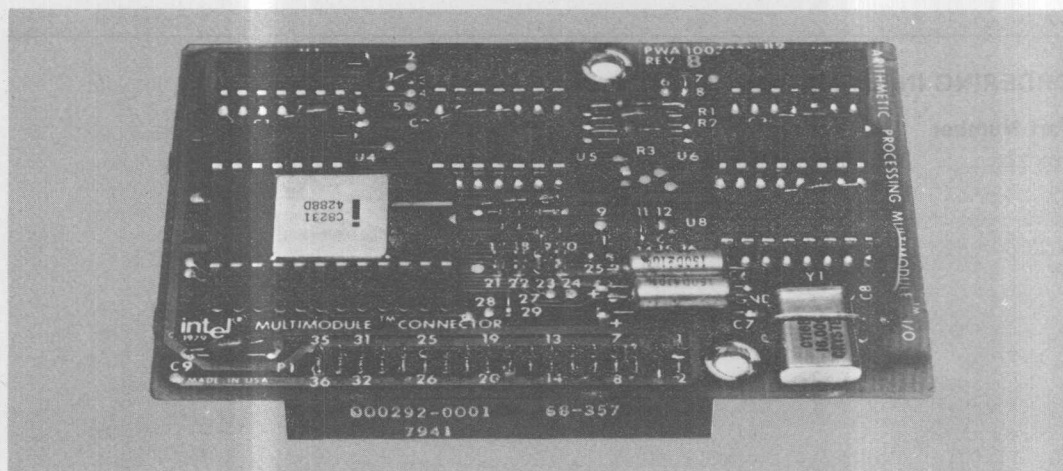
Part Number	Description
SBX 332	Floating Point Math MULTIMODULE Board



ISBX 331 FIXED/FLOATING POINT MATH MULTIMODULE BOARD

- iSBX bus compatible high speed fixed/floating point math expansion
- 4 MHz operation
- Fixed point single and double precision (16/32-bit)
- Floating point double precision (32-bit)
- Binary data formats
- Add, subtract, multiply and divide
- Trigonometric and inverse trigonometric functions
- Square root, log, and exponential functions
- Float-to-fixed and fixed-to-float conversions
- End of operation interrupt
- Software reset control
- Low power requirements
- iSBX bus on-board expansion eliminates MULTIBUS system bus latency and increases system throughput

The Intel® iSBX 331 Fixed/Floating Point Math MULTIMODULE Board is a member of Intel's new line of iSBX bus compatible MULTIMODULE products. The iSBX MULTIMODULE board plugs directly into any iSBX bus compatible host board offering low cost incremental on-board expansion. As a result, any iSBX bus compatible host board may be expanded to perform high speed math computations, affording up to a 40x improvement in speed compared to software math. The iSBX 331 module performs single/double (16/32-bit) precision fixed point plus double (32-bit) precision floating point arithmetic operations. In addition, the module performs transcendental, data manipulation, and fixed to float/float to fixed point conversion operations. The command operations run entirely independent of the host board permitting efficient concurrent processing. The iSBX board is closely coupled to the host board through the iSBX bus, and as such, offers maximum on-board performance and frees MULTIBUS system traffic for other system resources. Incremental power dissipation is minimal requiring only 2.73 watts.



FUNCTIONAL DESCRIPTION

The iSBX 331 module uses the Intel 8231 Arithmetic Processing Unit (APU) to accomplish high speed (4 MHz) math operation. The system software may communicate with the iSBX 331 module across the iSBX bus using I/O read/write commands. All transfers, including operand, result, status, and command information, take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack and commands are issued to perform operations on the data. Results are then available from the stack. A status byte may be read to monitor execution completion and the nature of the result (zero, sign, or errors). In addition, control logic is included on the iSBX 331 module to facilitate single instruction software reset control.

Command Functions

The iSBX 331 module commands fall into three categories: double precision floating point, single precision fixed point, and double precision fixed point (see Table 1). There are four arithmetic operations that can be performed in either fixed or floating point numbers: add, subtract, multiply, and divide. These operations require two operands. The 8231 assumes these operands are located in the internal stack as Top of Stack (TOS) and Next on Stack (NOS). The result will always be

returned to TOS. There are four types of transcendental operations that can be performed in floating point numbers: trigonometric functions, logarithms, exponentials, and square roots. The results of these operations will be returned to TOS. There are four types of data manipulation operations that can be performed in either fixed or floating point numbers: sign change of TOS, exchange of TOS and NOS and copying or popping operands onto or off of TOS. Fixed to floating point conversion can be performed on floating point instructions and floating point to fixed point conversion can be performed on fixed point instructions.

The execution times of the commands are shown in Table 2.

Interrupt Requests

There is one interrupt line from the APU that may generate an interrupt request to the host: END (MINTRI). The END interrupt line is active upon command completion. The END signal is cleared by a reset or status register read.

Installation

The iSBX 331 module plugs directly into the female iSBX connector on the host board. The module is then secured at one additional point with nylon hardware to insure the mechanical security of the assembly (see Figures 1 and 2).

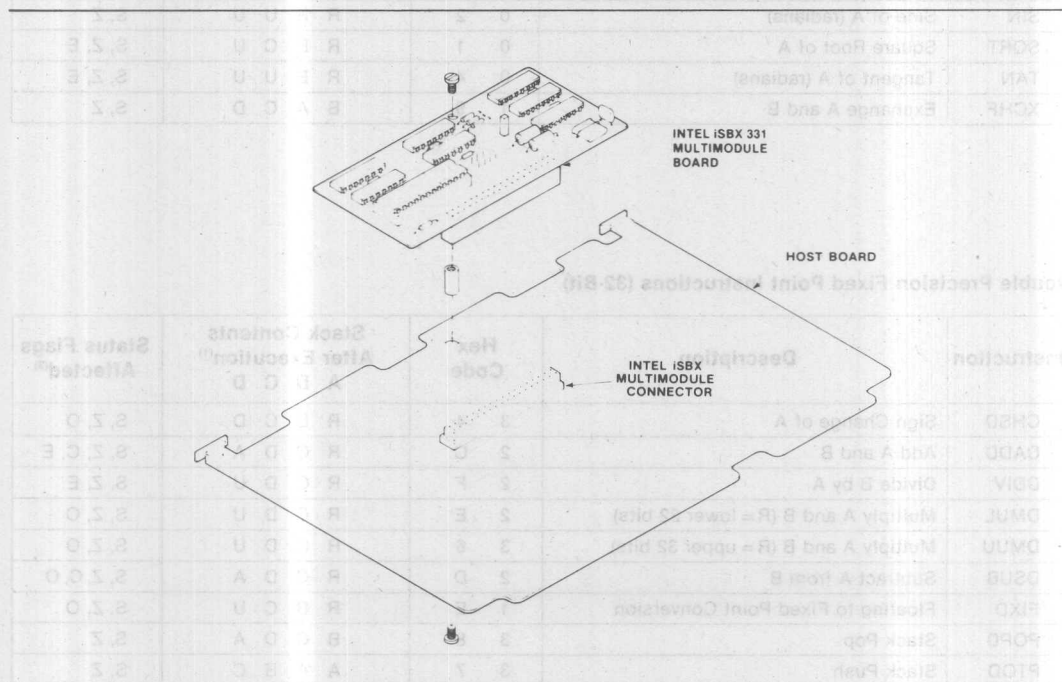


Figure 1. Installation of iSBX 331 Module on a Host Board

Table 1. Command Summary

Double Precision Floating Point Instructions (32-Bit)

Instruction	Description	Hex Code	Stack Contents After Execution ⁽¹⁾ A B C D	Status Flags Affected ⁽³⁾
ACOS	Inverse Cosine of A	0 6	R U U U	S, Z, E
ASIN	Inverse Sine of A	0 5	R U U U	S, Z, E
ATAN	Inverse Tangent of A	0 7	R B U U	S, Z
CHSF	Sign Change of A	1 5	R B C D	S, Z
COS	Cosine of A (radians)	0 3	R B U U	S, Z
EXP	e ^A Function	0 A	R B U U	S, Z, E
FADD	Add A and B	1 0	R C D U	S, Z, E
FDIV	Divide B by A	1 3	R C D U	S, Z, E
FLTD	32-Bit Fixed to Floating Point Conversion	1 C	R B C U	S, Z
FLTS	16-Bit Fixed to Floating Point Conversion	1 D	R B C U	S, Z
FMUL	Multiply A and B	1 2	R C D U	S, Z, E
FSUB	Subtract A from B	1 1	R C D U	S, Z, E
LOG	Common Logarithm (base 10) of A	0 8	R B U U	S, Z, E
LN	Natural Logarithm of A	0 9	R B U U	S, Z, E
POPF	Stack Pop	1 8	B C D A	S, Z
PTOF	Stack Push	1 7	A A B C	S, Z
PUPI	Push π onto Stack	1 A	R A B C	S, Z
PWR	B ^A Power Function	0 B	R C U U	S, Z, E
SIN	Sine of A (radians)	0 2	R B U U	S, Z
SQRT	Square Root of A	0 1	R B C U	S, Z, E
TAN	Tangent of A (radians)	0 4	R B U U	S, Z, E
XCHF	Exchange A and B	1 9	B A C D	S, Z

Double Precision Fixed Point Instructions (32-Bit)

Instruction	Description	Hex Code	Stack Contents After Execution ⁽¹⁾ A B C D	Status Flags Affected ⁽³⁾
CHSD	Sign Change of A	3 4	R B C D	S, Z, O
DADD	Add A and B	2 C	R C D A	S, Z, C, E
DDIV	Divide B by A	2 F	R C D U	S, Z, E
DMUL	Multiply A and B (R = lower 32 bits)	2 E	R C D U	S, Z, O
DMUU	Multiply A and B (R = upper 32 bits)	3 6	R C D U	S, Z, O
DSUB	Subtract A from B	2 D	R C D A	S, Z, C, O
FIXD	Floating to Fixed Point Conversion	1 E	R B C U	S, Z, O
POPD	Stack Pop	3 8	B C D A	S, Z
PTOD	Stack Push	3 7	A A B C	S, Z
XCHD	Exchange A and B	3 9	B A C D	S, Z

Table 1. Command Summary (continued)

Single Precision Fixed Point Instructions (16-Bit)

Instruction	Description	Hex Code	Stack Contents After Execution ⁽²⁾								Status Flags Affected ⁽³⁾
			A _U	A _L	B _U	B _L	C _U	C _L	D _U	D _L	
CHSS	Change Sign of A _U	7 4	R	A _L	B _U	B _L	C _U	C _L	D _U	D _L	S, Z, O
FIXS	Floating to Fixed Point Conversion	1 F	R	B _U	B _L	C _U	C _L	U	U	U	S, Z, O
POPS	Stack Pop	7 8	A _L	B _U	B _L	C _U	C _L	D _U	D _L	A _U	S, Z
PTOS	Stack Push	7 7	A _U	A _U	A _L	B _U	B _L	C _U	C _L	D _U	S, Z
SADD	Add A _U and A _L	6 C	R	B _U	B _L	C _U	C _L	D _U	D _L	A _U	S, Z, C, E
SDIV	Divide A _L by A _U	6 F	R	B _U	B _L	C _U	C _L	D _U	D _L	U	S, Z, E
SMUL	Multiply A _L by A _U (R = lower 16 bits)	6 E	R	B _U	B _L	C _U	C _L	D _U	D _L	U	S, Z, E
SMUU	Multiply A _L by A _U (R = upper 16 bits)	7 6	R	B _U	B _L	C _U	C _L	D _U	D _L	U	S, Z, E
SSUB	Subtract A _U from A _L	6 D	R	B _U	B _L	C _U	C _L	D _U	D _L	A _U	S, Z, C, E
XCHS	Exchange A _U and A _L	7 9	A _L	A _U	B _U	B _L	C _U	C _L	D _U	D _L	S, Z
NOP	No Operation	0 0	A _U	A _L	B _U	B _L	C _U	C _L	D _U	D _L	

NOTES:

1. The stack initially is composed of four 32-bit numbers (A, B, C, D). A is equivalent to Top Of Stack (TOS) and B is Next On Stack (NOS). Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A, B, C, or D).
2. The stack initially is composed of eight 16-bit numbers (A_U, A_L, B_U, B_L, C_U, C_L, D_U, D_L). A_U is the TOS and A_L is NOS. Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A_U, A_L, B_U, B_L,...).
3. Nomenclature: Sign (S); Zero (Z); Overflow (O); Carry (C); Error Code Field (E).

Table 2. Command Execution Times

Command Mnemonic	μSeconds	Command Mnemonic	μSeconds
SADD	4.25	ASIN	1917
SSUB	7.5	ACOS	1933.5
SMUL	21-23.5	ATAN	1501.5
SMUU	20-24.5	LOG	1118.5-1783
SDIV	21-23.5	LN	1074.5-1739
DADD	5.25	EXP	948.5-1219.5
DSUB	9.5	PWR	2072.5-3008
DMUL	48.5-52.5	NOP	1
DMUU	45.5-54.5	CHSS	5.75
DDIV	52	CHSD	6.75
FIXS	23-54	CHSF	4.5
FIXD	25-86.5	PTOS	4
FLTS	24.5-46.5	PTOD	5
FLTD	24.5-94.5	PTOF	5
FADD	13.5-92	POPS	2.5
FSUB	17.5-92.5	POPD	3
FMUL	36.5-42	POPF	3
FDIV	38.5-46	XCHS	4.5
SQRT	200	XCHD	6.5
SIN	1116	XCHF	6.5
COS	1029.5	PUPU	4
TAN	1438.5		

NOTE: Assumes 4 MHz operation.

ISBX 331

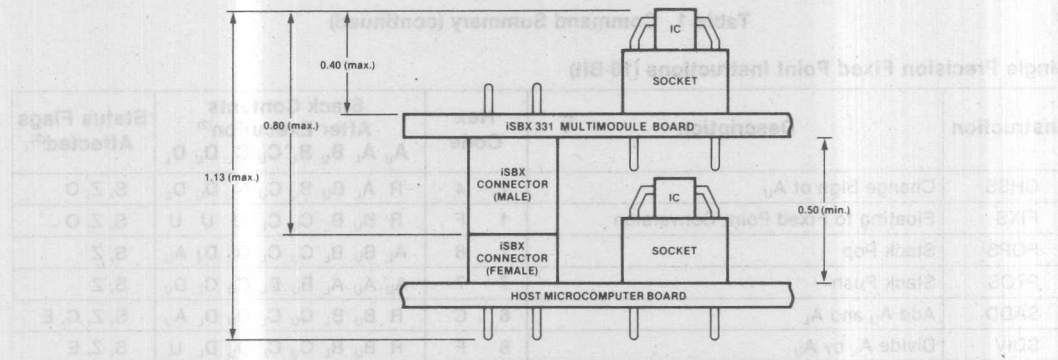


Figure 2. ISBX 331 MULTIMODULE BOARD Mounting Clearances (inches)

SPECIFICATIONS

Word Size

Data—8 bits.

On-Board Clock Rate

4.0 MHz \pm 0.1%.

I/O Addressing

Function	Type of Operation	iSBX Connector Port Address
Data Transfer	Read or Write	X0, X2, X4, or X6
Command Transfer	Write	X1, X3, X5, or X7
Status Transfer	Read	X1, X3, X5, or X7
Reset	Write	X8 through XF

NOTE:

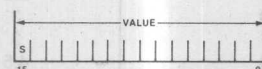
The port addresses are determined on the host iSBC microcomputer. Refer to the Hardware Reference Manual for your host iSBC microcomputer to determine the first digit (X) of the connector port addresses.

Arithmetic Functions

See Table 1.

Data Formats

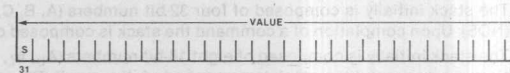
Single Precision Fixed Point (16 bits)



Bit 15: S = Sign of the operand. Positive values are represented by a sign bit of zero (S = 0). Negative values are represented by the two's complement of the corresponding positive value with a sign bit equal to 1 (S = 1).

Bits 0-14: Values in the range from - 32, 768 to + 32, 767.

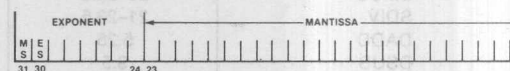
Double Precision Fixed Point (32 bits)



Bit 31: S = Sign of operand. Positive values are represented by a sign of zero (S = 0). Negative values are represented by the two's complement of the corresponding positive value with a sign bit equal to 1 (S = 1).

Bits 0-30: Values in the range from - 2, 147, 483, 648 to + 2, 147, 483, 647.

Double Precision Floating Point (32 bits)



Bit 31: MS = Sign of the mantissa. 1 represents negative and 0 represents positive.

Bits 24-30: ES = the exponent expressed as a two's complement 7-bit value having a range of - 64 to + 63.

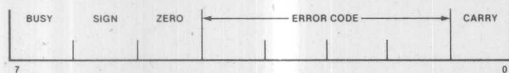
Bits 0-23: The mantissa is expressed as a 24-bit (fractional) value. The 8231 APU requires that floating point data be represented by a fractional mantissa value between 0.5 and 1 multiplied by 2 raised to an appropriate power (exponent). This is expressed as follows:

$$\text{Value} = \text{mantissa} \times 2^{\text{exponent}}$$

iSBX 331

Device Status

Device status is provided by means of an internal status register whose format is shown below:



BUSY: Indicates that 8231 is currently executing a command (1 = Busy)

SIGN: Indicates that the value on the top of stack is negative (1 = Negative)

ZERO: Indicates that the value on the top of stack is zero (1 = Value is zero)

ERROR CODE: This field contains an indication of the validity of the result of the last operation. The error codes are:

- 0000 — No error
- 1000 — Divide by zero
- 0100 — Square root or log of negative number
- 1100 — Argument of inverse sine, cosine, or e^x too large
- XX10 — Underflow
- XX01 — Overflow

CARRY: Previous operation resulted in carry or borrow from most significant bit. (1 = Carry/Borrow, 0 = No Carry/No Borrow.)

If the BUSY bit in the status register is a one, the other status bits are not defined; if zero, indicating not busy, the operation is complete and the other status bits are defined as given above.

Access Time

Read—1900 ns (max.)

Write—1900 ns (max.)

NOTE:

Actual transfer speed is dependent upon the cycle time of the host microcomputer. The listed times assume no operation in progress. If an operation is executing when an access is attempted, the command execution time must be added to the above times for all accesses except status read.

Interrupts

One interrupt request may originate from the APU indicating command completion (END).

Interface

iSBX Bus—All signals TTL compatible

Physical Characteristics

Width—6.35 cm (2.50 in.)

Length—9.40 cm (3.70 in.)

Height*—2.04 cm (0.80 in.) iSBX 331 Board

—2.86 cm (1.13 in.) iSBX 331 Board +
Host Board

Weight—51 gm (1.79 oz)

*See Figure 2.

Electrical Characteristics

DC Power Requirements

$V_{CC} = +5V \pm 5\%$ $I_{CC} = 365 \text{ mA max.}$

$V_{DD} = +12V \pm 5\%$ $I_{DD} = 75 \text{ mA max.}$

Environmental

Operating Temperature—0°C to 55°C

Free moving air across the base board and iSBX board.

Reference Manual

142668-01—iSBX 331 Floating Point Math
MULTIMODULE Board (NOT SUPPLIED)

Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBX 331	Fixed/Floating Point Math MULTIMODULE Board

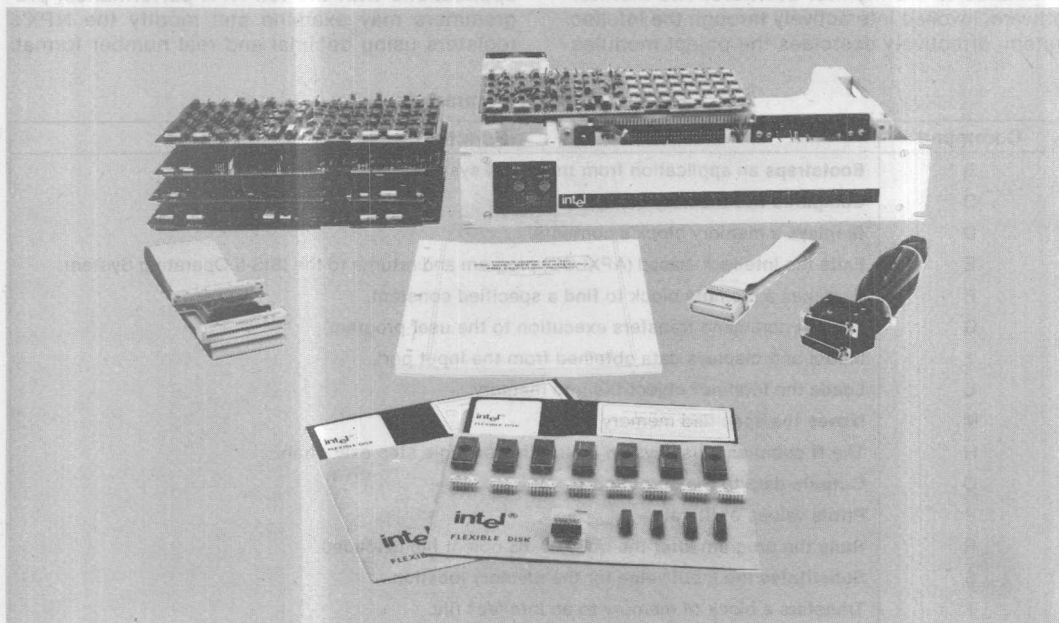


iSBC™ 957B

IAPX 86, 88 INTERFACE AND EXECUTION PACKAGE

- Supports target system debugging for iSBC™ 86/05, 86/12A, 88/25, 88/40 or iAPX 86, 88-based applications
- Interactively extends the Intellec® development environment to the target system for code execution and symbolic displays of results
- Supports custom and iRMX™ operating systems with application access to ISIS-II files
- Supports the 8087 Numeric Processor Extension (NPX) functions for high-speed arithmetic applications
- Utilizes a parallel or serial connection between the Intellec® Development System and the target system
- Provides an applications bootstrap from iRMX™ 86 and 88 file compatible peripherals

The Intel iSBC 957B package contains the necessary hardware, software, cables, terminator packs and documentation required to interface, through a serial or parallel connection, an iSBC 86/05, 86/12A, 88/25, 88/40 or iAPX 86, 88 target system to an MDS 800, Series II or Series III Intellec Microcomputer Development System for full-speed execution and debugging of application software. The iSBC 957B package supports the OEM's choice of a custom operating system, iRMX 86 Operating System or iRMX 88 Real-Time Multitasking Executive for the target application system. OEM's may utilize any iRMX 86, 88 supported target system peripheral for a bootstrap of the application system or have full access to the ISIS-II files of the Intellec system.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and ICS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

FUNCTIONAL DESCRIPTION

Overview

Extending the software development capabilities of the Intellec Microcomputer Development System, the iSBC 957B, iAPX 86, 88 Interface and Execution Package provides a link to executing and debugging in a target system. The application software, developed under the Intellec-resident ISIS-II Operating System, can readily be downloaded to an iSBC 86, 88 Single Board Computer or a custom iAPX 86, 88 system using the included monitor and its powerful, interactive debugging commands via the Intellec console. Programmers can effectively develop applications to ensure timely product availability.

Target System Debugging

The iSBC 957B package includes a communications link and target system resident monitor software for target application debugging. The target system monitor supports debugging through an attached CRT or the Intellec System. The Intellec resident communications software manages the link (serial or parallel) between the Intellec system and the target system. The communications software passes the appropriate console-requested commands to the monitor software. The monitor software, invoked interactively through the Intellec system, effectively exercises the object modules

on the target system. Pre-configured EPROM resident monitors are supplied for the iSBC 86/05, 86/12A and 88/40 products. The monitor software is configurable as to the selection of the communication link, processor board, numeric processor extension and the bootstrap loader functions. The OEM would burn the configured monitor into EPROMs for the target system.

The execution command environment (see Table 1) supported by the resident monitor loads object code into memory, executes it at full speed, sets breakpoints and examines the results. The monitor selectively executes portions of program modules based on breakpoints and single stepping requests. The monitor also provides commands to examine memory, manage memory movement by block, search for values, compare contents, and modify its value. Other program debugging information is provided through the displaying, examining or modifying of the iAPX 86, 88 registers.

Numeric Data Processor Support

Arithmetic applications utilizing the 8087 Numeric Processor Extension (NPX) are fully supported by the iSBC 957B Monitor. In addition to executing applications with the full NPX performance, programmers may examine and modify the NPX's registers using decimal and real number format.

Table 1. Monitor Commands

Command	Function
B	Bootstraps an application from the target system's peripheral device.
C	Compares two memory blocks.
D	Displays a memory block's contents.
E	Exits the Intellec®-based (APXLOD) program and returns to the ISIS-II Operating System.
F	Searches a memory block to find a specified constant.
G	The GO command transfers execution to the user program.
I	Inputs and displays data obtained from the input port.
L	Loads the Intellec® object file into memory.
M	Moves the specified memory block.
N	The N command displays an instruction's single step execution.
O	Outputs data to the output port.
P	Prints values of literals.
R	Runs the program after the iAPX 86, 88 object file is loaded.
S	Substitutes the input value for the memory location.
T	Transfers a block of memory to an Intellec® file.
X	Allows iAPX™ 86, 88 or NPX registers to be examined or modified.
*	Indicates remainder of the line is a comment.

The programmer feels confident that correct and meaningful numbers are entered for the application without having to encode and decode complex real, integer, and BCD hexadecimal formats.

Easy-To-Use Connection

The physical interface between the Intellect Microcomputer Development System and the target iAPX 86, 88 system is accomplished with the supplied iSBC 957B cables. The cabling arrangement is either a serial or parallel line and varies depending on whether the development system is of the Intellect MDS 800 family or one of the Intellect Series II or III family. All communication, including

data transfer and command requests, occurs over this line.

As shown in Figure 1, the connection to the target iSBC 86/12A system is accomplished with the iSBC 86/12A serial port through an RS232C serial line interconnected to Serial port 1 of an Intellect Series II Model 220 or Model 230 or the CRT port of an Intellect 800 Development System. In some target iSBC application systems, the serial I/O port may not be available for debugging. The example connection, shown in Figure 2, shows how the parallel port of the iSBC 86/12A board is connected through the parallel cable to the UPP port of the Intellect system.

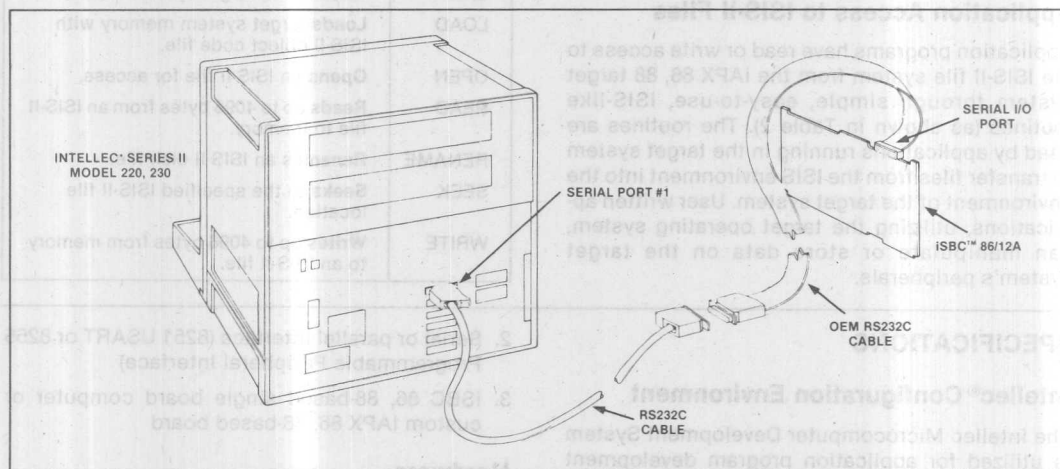


Figure 1. Intellect® Series Models 220, 230 Serial Connection

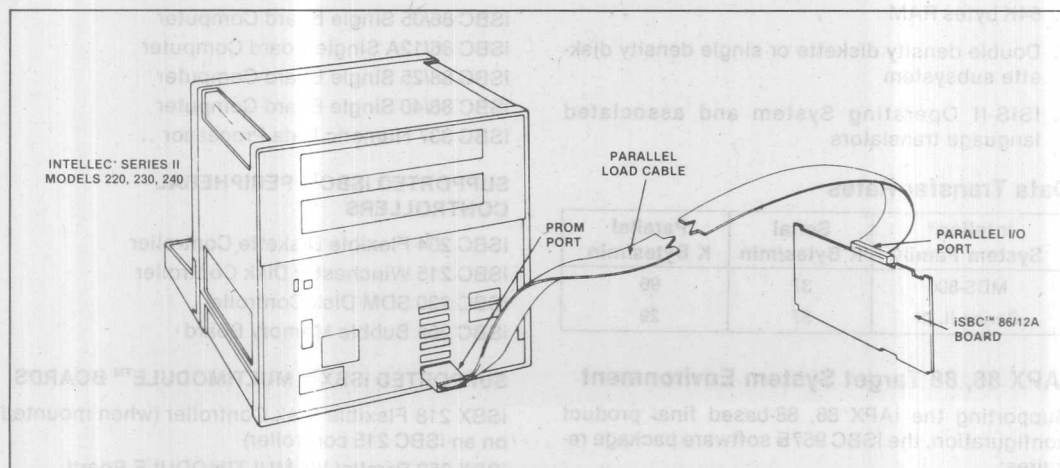


Figure 2. Intellect® Series Models 220, 230, 240 Parallel Connection

Application Bootstrap Loader

The bootstrap loader, invoked from a stand-alone CRT attached to the iSBC product or the Intellec console, dynamically loads the application system into the target system's memory from an attached peripheral through the iRMX 86, 88 compatible bootstrap loader. This configurable function can be included with the iSBC 957B monitor and installed in the iAPX 86, 88 target system. The programmer may load application code modules or an entire system from bubble memory, floppy diskettes, Winchester disks or other iRMX supported mass storage devices.

Application Access to ISIS-II Files

Application programs have read or write access to the ISIS-II file system from the iAPX 86, 88 target system through simple, easy-to-use, ISIS-like routines (as shown in Table 2). The routines are used by applications running in the target system to transfer files from the ISIS environment into the environment of the target system. User written applications, utilizing the target operating system, can manipulate or store data on the target system's peripherals.

SPECIFICATIONS

Intellec® Configuration Environment

The Intellec Microcomputer Development System is utilized for application program development and requires the following to support the iSBC 957B package:

1. 64K bytes RAM
2. Double density diskette or single density diskette subsystem
3. ISIS-II Operating System and associated language translators

Data Transfer Rates

Intellec® System Family	Serial K Bytes/min	Parallel K Bytes/min
MDS-800	37	96
Series II, III	37	29

iAPX 86, 88 Target System Environment

Supporting the iAPX 86, 88-based final product configuration, the iSBC 957B software package requires:

1. iSBC 957B cable and EPROMmed monitor

Table 2. Routines for ISIS-II Services Available to Target System Applications

Routine	Target System Function
ATTRIB	Changes an ISIS-II file attribute .
CI	Returns a character input from the console .
CLOSE	Closes an opened ISIS-II file.
CO	Transfers a character for console output .
DELETE	Deletes the specified ISIS-II file.
ERROR	Displays an error message on the Intellec® console.
EXIT	Exits to the target system monitor.
LOAD	Loads target system memory with ISIS-II object code file.
OPEN	Opens an ISIS-II file for access.
READ	Reads up to 4096 bytes from an ISIS-II file to memory.
RENAME	Renames an ISIS-II disk file.
SEEK	Seeks to the specified ISIS-II file location.
WRITE	Writes up to 4096 bytes from memory to an ISIS-II file.

2. Serial or parallel interface (8251 USART or 8255 Programmable Peripheral Interface)
3. iSBC 86, 88-based single board computer or custom iAPX 86, 88-based board

Hardware

SUPPORTED iSBC™ MICROCOMPUTERS

iSBC 86/05 Single Board Computer
iSBC 86/12A Single Board Computer
iSBC 88/25 Single Board Computer
iSBC 88/40 Single Board Computer
iSBC 337 Numeric Data Processor

SUPPORTED iSBC™ PERIPHERAL CONTROLLERS

iSBC 204 Flexible Diskette Controller
iSBC 215 Winchester Disk Controller
iSBC 220 SDM Disk Controller
iSBC 254 Bubble Memory Board

SUPPORTED iSBX™ MULTIMODULE™ BOARDS

iSBX 218 Flexible Disk Controller (when mounted on an iSBC 215 controller)
iSBX 350 Parallel I/O MULTIMODULE Board
iSBX 351 Serial I/O MULTIMODULE Board

iSBC™ 957B Package Contents (Supplied)

CABLES

- 1 — Serial I/O port of iSBC or iAPX 86, 88 compatible product to male RS232C connector
- 1 — Intellec System RS232C port to female RS232C connector
- 1 — Parallel load cable to mate between Intellec System UPP port and parallel I/O port on iSBC or iAPX 86, 88 compatible product

PARALLEL INTERFACE ADAPTER

- 1 — Parallel port status adapter for iSBC products using parallel cable

I/O DRIVER AND TERMINATORS

- 4 — iSBC 901 - 220 ohm/330 ohm terminator packs
- 4 — iSBC 902 - 1K ohm terminator packs
- 4 — 7437 line driver packs

INTERFACE AND EXECUTION SOFTWARE DISKETTES

- 1 — Single density, ISIS compatible
- 1 — Double density, ISIS compatible

SYSTEM MONITOR EPROMs

Microcomputer	Address		Supports NPX
	RAM	ROM	
iSBC™ 86/05, 86/12A	00000H-007FFH	FC000H-FFFFFFH	Yes
iSBC™ 88/40	00000H-006FFH	FD000H-FFFFFFH*	No

* Allows 2816 E2PROM to be used at FC000H

Reference Manual (Supplied)

143979-002 — User's Guide for the iSBC 957B, iAPX 86, 88 Interface and Execution Package

ORDERING INFORMATION

Part Number Description

SBC 957B Intellec to iAPX 86, 88 Interface and Execution Package including software, cables and EPROMs.

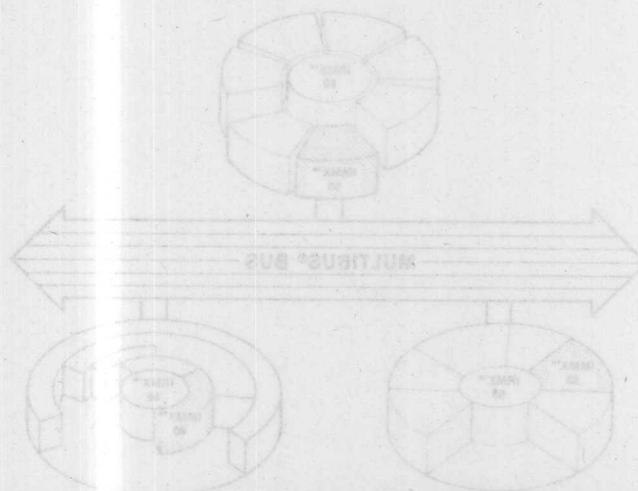


Figure 1. iSBC 957B Real-Time Executive Interface

The Intel Corporation and its subsidiaries are not responsible for the use of the information contained herein for any purpose other than that for which it was intended. The Intel Corporation and its subsidiaries are not responsible for the use of the information contained herein for any purpose other than that for which it was intended.



iMMX™ 800 MULTIBUS® MESSAGE EXCHANGE SOFTWARE

- Supports use of multiple processors on the MULTIBUS® system bus
- Increases total system throughput
- Implements Intel-standard multi-processing protocol
- Supports combination of 8- and 16-bit boards in one design
- Helps solve critical response-time problems
- Includes Ethernet* device driver
- Provides hardware-independent application interface
- Supports iRMX™ 80, iRMX™ 86, and iRMX™ 88 applications

The iMMX MULTIBUS Message Exchange provides an Open Multiprocessing System. It allows tasks-executing on separate processors to communicate by sending messages. By providing an off-the-shelf implementation of the MULTIBUS Interprocessor Protocol, it cuts many man-months off the typical development schedule. Loosely coupled multiprocessing makes multiple-microcomputer applications simple: programs request message transfer by means of a small set of systems calls — the iMMX software takes care of providing reliable message transfer via shared MULTIBUS memory.

The iMMX product is open to high-performance applications, encourages modular design practices, and supports multiple operating systems. By making it easy to use multiple processors it increases total system throughput and allows processing power to be optimized for both I/O handling and data processing. Once an initial application design is complete, it can be easily enhanced by adding new tasks and/or processors. The iMMX product allows the engineer to choose from a wide range of 8- and 16-bit iSBC microcomputers — from the iSBC 80/24 board to the iSBC 86/30 single board computer. The net result is a combination of performance and flexibility that meets the needs of a diverse set of multiple-microcomputer applications.

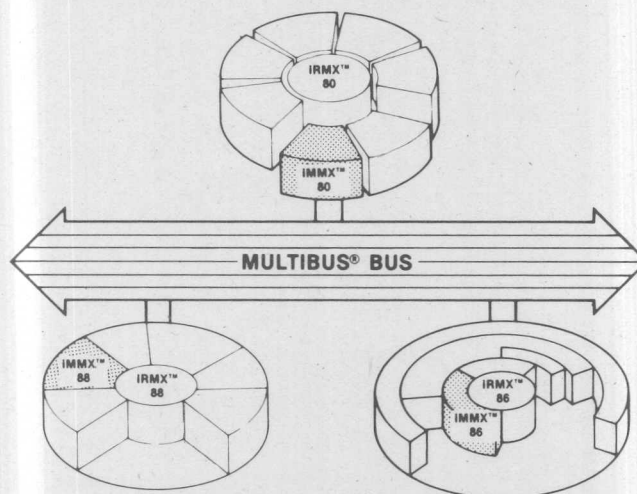


Figure 1. iMMX 800 Real-Time Executive Interface

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. * ETHERNET is a trademark of Xerox Corporation.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Open Multiprocessing System

OPEN TO HIGH PERFORMANCE APPLICATIONS

The iMMX supports high performance applications in two ways. First, it increases the total system throughput by allowing multiple processors to be easily incorporated in an application. Second, critical response time requirements can be met by placing computing power close to each critical input. Application programmers can concentrate on added-value functions while iMMX software takes care of variable length transfers, shared memory management, mutual exclusion, interprocessor interrupts, and hardware details.

OPEN TO MODULAR DESIGN

By supporting modular design, iMMX software provides four key benefits. First, each hardware module can be selected or designed according to needs of a subsystem; the iMMX 800 software takes care of the integration. Second a whole range of products can be created from a few hardware/software modules. Third, the breadth of products available for the industry-standard MULTIBUS dramatically reduces the amount of custom-design work required to complete a system. Finally, as customers, new markets, or competition re-

quires, performance can be enhanced or new features can be added by adding new modules.

OPEN TO MULTIPLE OPERATING SYSTEMS

iMMX software supports both standard Intel iRMX operating systems and custom systems. Off-the-shelf support is provided for iRMX 80, iRMX 86, and iRMX 88 applications — allowing the engineer to choose the best match for each problem. In addition, the underlying MULTIBUS Interprocessor Protocol (MIP) is completely specified so that custom operating systems and other subsystems can be integrated with iRMX-based subsystems.

Loosely-Coupled Multiprocessing

The iMMX 800 software supports loosely-coupled multiprocessor systems. The software interface is composed of simple, easy-to-use, modules. By supporting the addressing, data transfer, control, and memory management functions, the software as shown in Figure 2, divides the operation into three functions: the virtual interface, the logical protocol, and the physical protocol.

The **virtual interface** is the application task's access to the iMMX services. Using this interface, a task can request a connection to a particular port. Using the connection, the task can request that messages be transferred to the task(s) that are requesting messages from the same port.

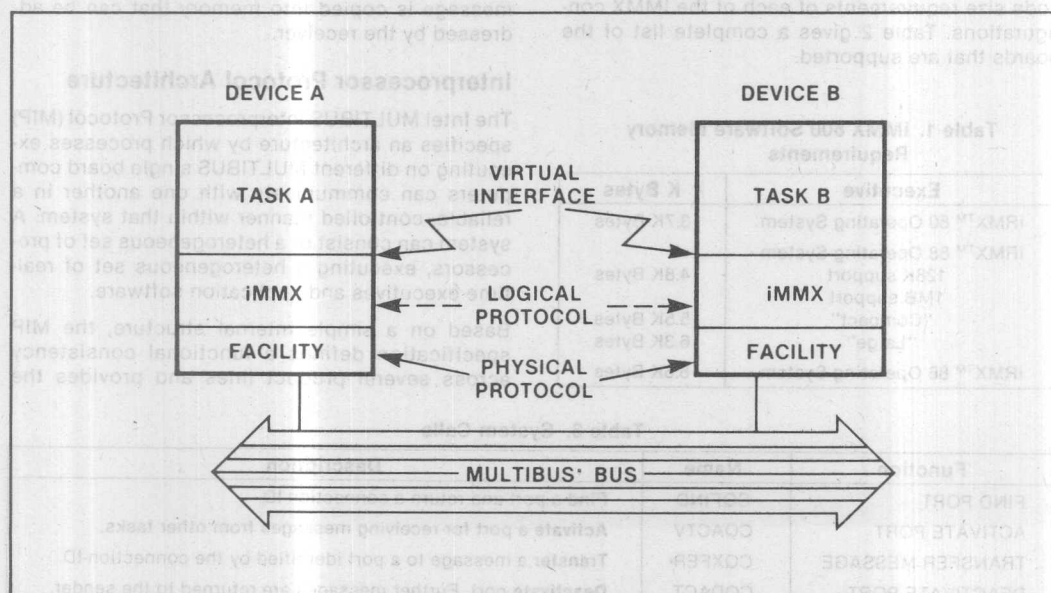


Figure 2. Inter-Device Task-to-Task Communications

The **logical protocol** supports a message manager function. The Message Manager prepares the message for delivery to a specific destination port based on the connection specified. In addition, the logical protocol returns status information about the transfer.

The **physical protocol** is implemented by VLSI and associated circuitry. This level of protocol includes data flow control, mutual exclusion mechanics, address recognition and interactive signalling requirements.

iMMX software supports four different inter-device signalling mechanisms: MULTIBUS interrupts, memory-mapped interrupts, I/O-port-mapped interrupts and polling.

iRMX™ Uniform Interface

The iMMX 800 software provides a uniform interface across all iRMX based software environments. The iMMX software services are provided as a set of tasks, system procedures, and interrupt drives.

Support is supplied for the iAPX 86/88-based microcomputers that support the iRMX 86 and the iRMX 88 Operating Systems. In addition, software support is provided Intel 8085-based products via the iRMX 80 Operating System. Table 1 shows the code size requirements of each of the iMMX configurations. Table 2 gives a complete list of the boards that are supported.

Table 1. iMMX 800 Software Memory Requirements

Executive	K Bytes
iRMX™ 80 Operating System	3.7K Bytes
iRMX™ 88 Operating System	
128K support	4.8K Bytes
1MB support	
"Compact"	5.5K Bytes
"Large"	6.3K Bytes
iRMX™ 86 Operating System	6.6K Bytes

Table 2. Supported Single Board Computers

iRMX™ 80 Operating System	iRMX™ 88 Operating System	iRMX™ 86 Operating System
iSBC® 80/24	iSBC® 86/05	iSBC® 86/05
iSBC® 80/30	iSBC® 86/12A	iSBC® 86/12A
iSBC® 544	iSBC® 86/14	iSBC® 86/14
iSBC® 569	iSBC® 86/30	iSBC® 86/30
	iSBC® 88/25	iSBC® 88/25
	iSBC® 88/40	iSBC® 88/40
	iSBC® 88/45	iSBC® 88/45

Message Transfer Mechanism

iMMX multiprocessing is based on a message-passing model. Tasks on each processor communicate with each other by sending and receiving messages to and from ports.

Table 3 shows five iMMX system calls: Find Port, Activate Port, Transfer Message, Deactivate Port and Lose Port.

Shared Memory Space

The iMMX software manages the message passing in such a way that a task that receives a message can address it even if the message originated in the private memory of another processor. This means that, when appropriate, the message is copied into memory that can be addressed by the receiver.

Interprocessor Protocol Architecture

The Intel MULTIBUS Interprocessor Protocol (MIP) specifies an architecture by which processes executing on different MULTIBUS single board computers can communicate with one another in a reliable, controlled manner within that system. A system can consist of a heterogeneous set of processors, executing a heterogeneous set of real-time executives and application software.

Based on a simple internal structure, the MIP specification defines a functional consistency across several product lines and provides the

Table 3. System Calls

Function	Name	Description
FIND PORT	CQFIND	Find a port and return a connection-ID.
ACTIVATE PORT	CQACTV	Activate a port for receiving messages from other tasks.
TRANSFER MESSAGE	CQXFER	Transfer a message to a port identified by the connection-ID.
DEACTIVATE PORT	CQDACT	Deactivate port. Further messages are returned to the sender.
LOSE	CQLOSE	Loses a connection to a port.

means to support efficient operation in multiple processor environments.

Ethernet Device Driver

The iMMX 800 package provides an iSBC 550 Ethernet Communications Controller device driver. This device driver uses iMMX routines to communicate to the iSBC 550 controller (see Figure 3). This same approach can be used to write other iRMX 88 and 86 device drivers.

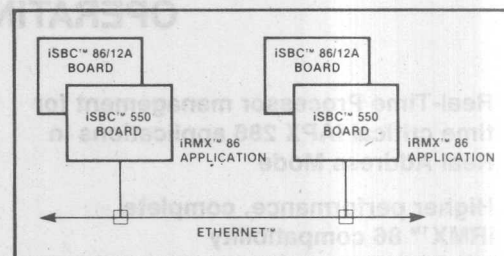


Figure 3. Ethernet Communications

SPECIFICATIONS

iSBC™ Supported Hardware

SINGLE BOARD COMPUTERS

iSBC 80/24
iSBC 80/30
iSBC 86/05
iSBC 86/12A
iSBC 86/14
iSBC 86/30
iSBC 88/25

iSBC 88/40
iSBC 88/45

INTELLIGENT CONTROLLERS

iSBC 544 (Communications)
iSBC 569 (Digital)
iSBC 550 (Communications)
via Ethernet driver

Reference Manual (Supplied)

iMMX 800 MULTIBUS Message Exchange
Reference Manual

ORDERING INFORMATION

Description

The iMMX 800 MULTIBUS Message Exchange Software is a licensed product that provides users of Intel Single Board Computers using the iRMX 80, iRMX 86, and iRMX 88 Operating Systems a standardized, memory-based, task-to-task communication protocol. This protocol provides the fundamental capabilities needed to exchange data between multiple 8-bit and 16-bit microcomputers residing on the same MULTIBUS system bus.

Part Number Description

MMX 800 ARO	Single Density Media. Requires incorporation fee for each derivative work.
MMX 800 BRO	Double Density Media. Requires incorporation fee for each derivative work.

MMX 800 ABY	Single Density Media. Includes incorporation fee buyout.
MMX 800 BBY	Double Density Media. Includes incorporation fee buyout.
MMX 800 AWX	Single Density Media. Update service for an additional year.
MMX 800 BWX	Double Density Media. Update service for an additional year.
MMX 800 LST	Human readable source listings for the iMMX 800 software modules.
MMX 800 LWX	Extends source listing updates for an additional year.

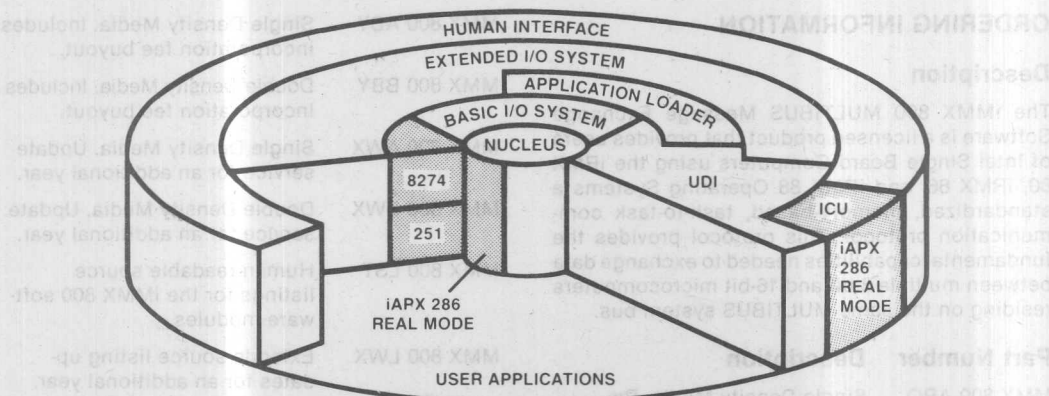
* ETHERNET is a trademark of Xerox Corporation.

iRMX™ 286R OPERATING SYSTEM

- Real-Time Processor management for time critical iAPX 286 applications in Real Address Mode
- Higher performance, complete iRMX™ 86 compatibility
- Complete support of 80287 Processor Extension
- New terminal driver for 8274 Multi Protocol Serial Controller (MPSC)
- New iSBC® 251 Bubble Memory Driver
- Multi-terminal support with multi-user human interface
- On-target system development with Universal Development Interface (UDI)
- Configurable system size and function for diverse application requirements
- All iRMX™ 286R code can be (P)ROM'ed to support totally solid state designs
- Powerful utilities for interactive configuration and real-time debugging
- Functions in conjunction with iRMX™ 86 Release 5

The high performance iRMX™ 286R Operating System, functioning in conjunction with the iRMX 86 Release 5 Operating System software, is designed to manage and extend the resources of iSBC® 286 Single Board Computers as well as other iAPX 286-based Microcomputers in Real Address Mode. The iRMX 286R Operating System is an easy-to-use, real-time, multi-tasking and multi-programming software system providing the capability of executing the entire configurable layers of the iRMX 86 software in the Real Address Mode of the iAPX 286 and the iSBC 286/10 Single Board Computer.

The new features added to this incremental System include drivers for both channels of an 8274 and a driver for the iSBX™ 251 MULTIMODULE™ board.



NEW IN
iRMX™ 286R



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, Multichannel and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supercedes previously published specifications on these devices from Intel.

© INTEL CORPORATION, 1983

5-10

April, 1983
Order Number: 210898-001

The iRMX™ 286R Operating System is a complete set of system software modules that provide resource management functions needed by most computer systems. These management functions are currently available to OEMs in Release 5 of the iRMX 86 Operating System. For a complete description of the functions, their value and performance, please refer to the Release 5 iRMX 86 Data Sheet (order number 210885-001).

This data sheet describes the new features provided by the iRMX 286R Operating System. These new features add new capabilities specially designed for OEMs wishing to take better advantage of the speed and performance of the recent VLSI microprocessor, the iAPX 286.

New drivers provided with the iRMX 286R Operating System include:

- An 8274 terminal device driver, to support the 8274 for two channel Async support of the iSBC® 286/10 board in RS232 mode.
- Support for the iSBX™ 251 Bubble Memory System, installed as a custom driver.

The 8274 terminal device driver supports two serial channels used in RS232 mode and supports all features of the iRMX 86 terminal support. To use the RMX debugger an iSBX 351 MULTIMODULE™ is required.

The iRMX 286R Operating System is a natural extension, growth path for iRMX 86 users offering a higher-performance, simple upgrade. The Interactive Configuration Utility has been enhanced to allow ease of configuration of iAPX 286 based microprocessors such as the iSBC 286/10 single board computer.

FUNCTIONAL DESCRIPTION

To take best advantage of the iAPX 286 microprocessor in applications where computers are required to perform many functions simultaneously, the iRMX 286R Operating System provides a multi-programming environment in which many independent, multi-tasking application programs may run. Each application environment may be treated separately to allow application programmers the flexibility to separately manage each application's resources while developing and testing the software for each independently.

The resource management functions of the iRMX 286R System are supported by a number of configurable software layers. While many of the functions supplied by the innermost layer, the Nucleus, are required by all systems, all other functions are optional. The I/O systems, for example, need not be included in systems with no secondary storage requirement. Each layer provides functions that encourage application programmers to maintain an understandable structure and to develop easily maintainable programs more quickly.

The components of the iRMX 286R Operating System provide both implicit and explicit management of a system's resources. These resources often include the processor's time and registers, the 80287 Numeric Data Processor Extension, up to one megabyte of system memory, up to 57 independent interrupt sources, all input and output devices, as well as directory and data files contained on mass storage devices. The Operating System makes it easier for designers to include many terminals in their systems by providing multi-terminal and multi-user support software, improved I/O performance, an Interactive Configuration Utility, and a sophisticated Crash Analyzer.

SPECIFICATIONS

Supported Software Products

iRMX 860	iRMX 86 Development Utilities Package including the iAPX 86 and 88 Linker, Locator, Macro Assembler, Librarian, and the iRMX 86 Editor
iRMX 861	PASCAL 86/88 Compiler
iRMX 862	FORTTRAN 86/88 Compiler
iRMX 863	PL/M 86/88 Compiler
iRMX 864	TX-Screen-Oriented Editor
iMMX 800	MULTIBUS® Message Exchange software package for iRMX 80, 86, and 88 application systems

Supported Hardware Products

COMPONENTS

iAPX 80286 Microprocessor
80287 Numeric Processor Extension
iAPX 86 and 88 Microprocessors
8087 Numeric Processor Extension
80130 Operating System Firmware Component
8253 and 8254 Programmable Interval Timers
8259A Programmable Interrupt Controller
8251A USART
8255 Programmable Parallel Interface
8274 USART 2 Channel (Serial) Controller

iSBC® MULTIBUS® PRODUCTS

iSBC 86/12A, 86/05, 86/14, 86/30, 88/25, 88/40, and 286/10 Single Board Computers

iSBC 204 Flexible Disk Controller

iSBC 206 Hard Disk Controller

iSBC 208 Flexible Disk Controller

iSBC 215 Winchester Disk Controller

iSBC 220 SMD Disk Controller

iSBC 251 Bubble Memory System

iSBC 254 Bubble Memory System

iSBC 534 4-Channel Terminal Interface

iSBC 544 Intelligent 4-Channel Terminal Interface and Controller

iSBX 218 Flexible Disk Controller

iSBX 350 Parallel Port (Centronix-type Printer Interface)

iSBX 351 Serial Communications Port

iSBX 270 CRT, Light Pen and Keyboard Interface

Available Literature

iRMX 286R Operating System Installation and Configuration Guide for Release 1 (145556-001)

All of the manuals listed below are supplied with iRMX 86 Release 5 and are available separately under the order numbers shown.

Introduction to the iRMX 86 Operating System (9803124-04)

iRMX 86 Operator's Manual (144523-001)

Master Index for iRMX 86 Release 5 Documentation (145015-001) (Not available until 1983)

Getting Started With The Release 5 iRMX 86 System (145073-001)

iRMX 86 Installation Guide (9803125-05)

iRMX Configuration Guide (9803126-05)

iRMX 86 Nucleus Reference Manual (9803122-04)

iRMX 86 Terminal Handler Reference Manual (143324-002)

iRMX 86 Debugger Reference Manual (143323-002)

iRMX 86 Basic I/O System Reference Manual (9803123-05)

iRMX 86 Loader Reference Manual (143318-002)

iRMX 86 Extended I/O System Reference Manual (143308-002)

iRMX 86 Human Interface Reference Manual (9803202-003)

Guide to Writing Device Drivers for the iRMX 86 and iRMX 88 I/O Systems (142926-004)

iRMX 86 Programming Techniques (142982-003)

User's Guide for the iSBC 957B, iAPX 86, 88 Interface and Execution Package (143979-002)

iRMX 86 Disk Verification Utility Reference Manual (144133-002)

Runtime Support Manual for iAPX 86, 88 Applications (121776-002)

iRMX 86 Crash Analyzer Reference Manual (144522-001)

ORDERING INFORMATION

The iRMX 286R facilities described in this data sheet require the iRMX 86 R5 Operating System as a prerequisite.

For all new iRMX users, there are a number of different licensing and support options available. All options are provided on either single or double density ISIS-formatted diskettes, on 5¼" iRMX 86-formatted diskettes, or on double density iRMX 86-formatted diskettes. ISIS-format diskettes may be used on Intel Intellec® Development Systems. The iRMX 86-format may be used on any iRMX 86-based system supporting the appropriate compilers and development environment.

The OEM license options listed here allow users to incorporate the iRMX 286R Operating System into their

applications. Each use requires payment of an Incorporation Fee.

Other licensing options include prepayment of future incorporation fees, single use rights for a single machine, support at a second development site, and one-year support service extensions.

Each option includes 90 days of update service that provides Software Problem Report Service and copies of System updates that occur during this period. All initial licenses include the iSDM™ 286 System Debug Monitor and iRMX 286R documentation.

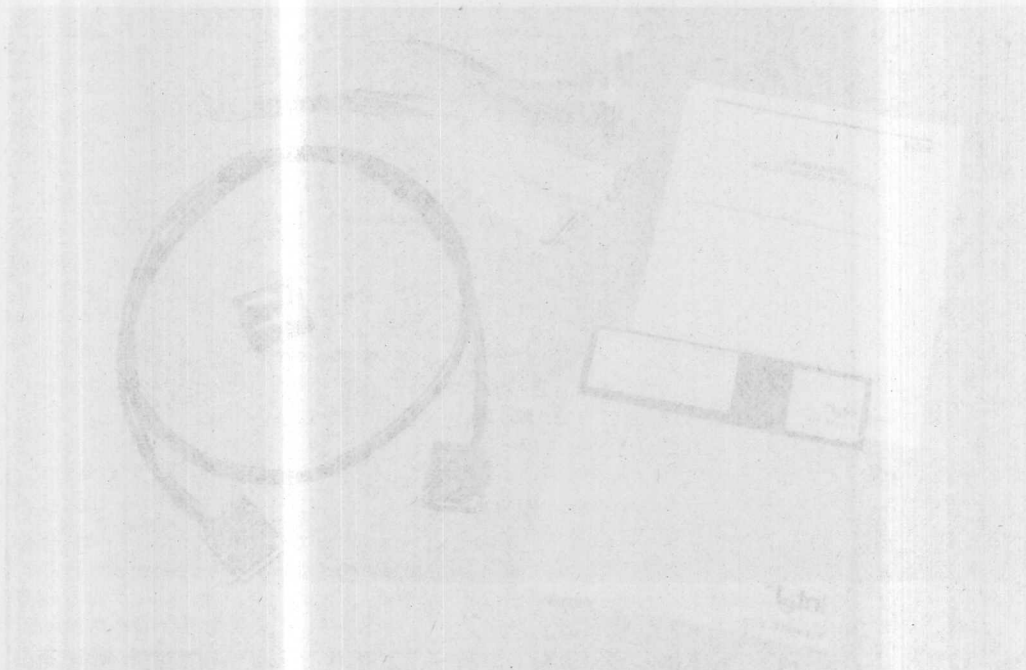
As with all Intel Software, purchase of any of these options requires execution of a standard Intel Master Software License.

Part Number	Description
RMX 286R KIT ARO	Single Density, OEM License
RMX 286R KIT BRO	Double Density, OEM License

RMX 286R KIT ERO Double Density, iRMX 86-format, OEM License for use on iRMX 86-based environments

NOTE: Each option requires previous purchase of iRMX 86 Release 5. iRMX 286R does not support the Protected Virtual Address Mode (PVAM) of the iAPX 286-based microprocessor.

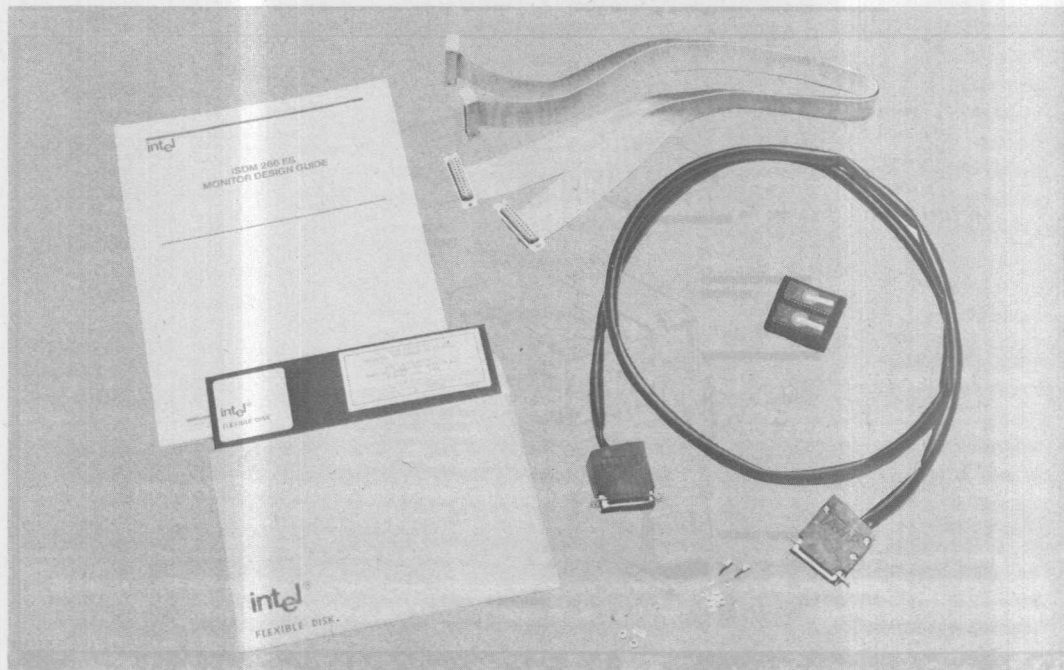
The iRMX™ 286R System Debug Monitor package contains the necessary hardware, software, cables, PROMs, and documentation required to interface an iSBCT™ 286 board or iAPX 286 component applications to an iRMX™ 286R Real-Time Debugging System or custom operating system, with debugging tools to examine CPU registers, memory, interrupt, CPU descriptor tables, and other crucial environmental details. The Monitor also allows programs to access files on the development system via the internal UDI support and the serial communication link.



iSDM™ 286 iAPX 286 SYSTEM DEBUG MONITOR

- Development support of iSBC® 286-and iAPX 286-based applications
- Real Address Mode (RAM) and Protected Virtual Address Mode (PVAM) support
- Universal Development Interface (UDI) support via development system connection
- Underlying debugging tool for iRMX™ 286R applications
- Supports 80287 Numeric Processor Extension (NPX) for high-speed math applications
- Program load capability from Intellec® Series III Development Systems
- Bootstrap Loader for iRMX™ 286R, 86, and 88 file compatible peripherals
- iAPX 286 single step operation allowed

The Intel iSDM™ 286 System Debug Monitor package contains the necessary hardware, software, cables, PROMs, and documentation required to interface an iSBC® 286 board or iAPX 286 component applications to an Intellec® Series III through a high-speed link. The System Debug Monitor supports an OEM's choice of the iRMX™ 286R Real-Time Multitasking Operating System or custom operating system, with debugging tools to examine CPU registers, memory content, CPU descriptor tables, and other crucial environmental details. The Monitor also allows programs to access files on the development system via the internal UDI support and the serial communication link.



FUNCTIONAL DESCRIPTION

Overview

The iSDM 286 System Debug Monitor provides programmers of iAPX 286-based applications with the debugging tools needed to test new applications ranging from single-user systems to complex operating systems. Programmers are given direct access to both the Real Address (ram) and Protected Virtual Address (pvam) Modes of the CPU via a simple terminal interface, or via an Intellec Series III Development System.

Universal Development Interface

Any iRMX 86, Series III, or other UDI-based application can be supported by the iSDM 286 Monitor. The Monitor emulates many of the UDI calls (ram or pvam), and passes all requests for a file system to the host development station. UDI applications such as compilers and other programs available from Independent Software Vendors can be tested in the target iAPX 286 environment immediately.

Powerful Debugging Commands

A powerful set of user functions includes commands to:

- Examine and Modify CPU Registers
- Examine, Modify, and Move memory locations
- Symbolic reference to variable names
- Find and compare memory contents
- Set program breakpoints
- Bootstrap load application software

Single-step CPU operation

Change between Real Address Mode and Protected Virtual Address Mode

Formatted Displays

The iSDM 286 Monitor formats all iAPX 286 pre-defined data structures into clearly understandable displays. This display gives programmers a formatted view of CPU registers such as LDTs, GDTs, IDTs, Segment Selectors, and Task State Segments — not just a series of unconnected digits.

Numeric Data Processor Support

In addition to executing 80287 Numeric Processor Extension (NPX) applications with full NPX performance, programmers may examine and modify NPX registers using decimal and real number format. Any location in memory known to contain numeric values in standard real format (IEEE P754) may be examined or modified using normal decimal notation. In this manner programmers may feel confident that correct and meaningful numbers are available to applications without having to encode and decode complex real, integer, and BCD hexadecimal formats.

High-Speed Serial Connection

Target application hardware is connected to the development system via a serial link capable of 19.2K baud. All control operations and UDI file manipulations occur over this link through the cables supplied. As shown in Figure 1, the serial link is supported by the iSBC 86 USART port of the development system.

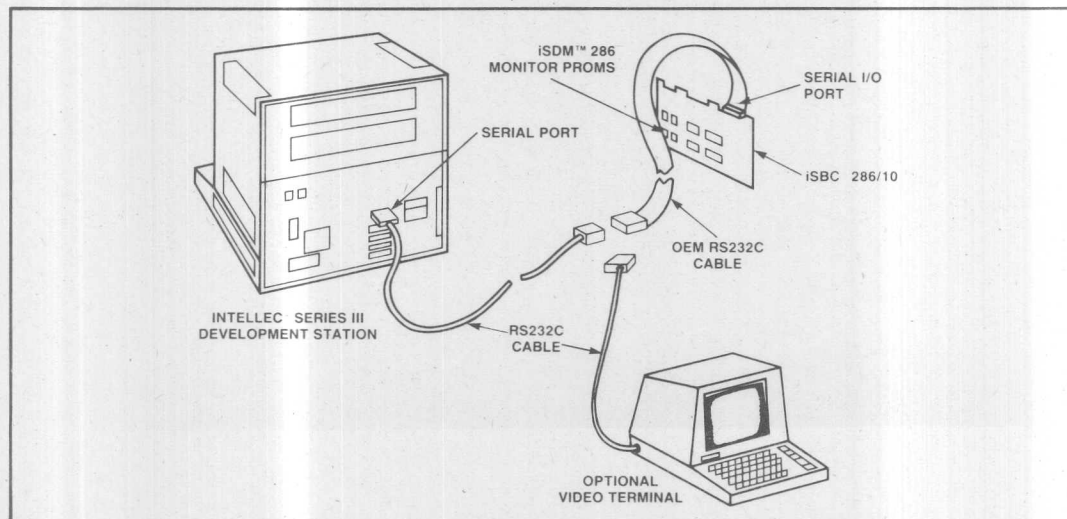


Figure 1. Typical iSDM™ 286 Environment

SPECIFICATIONS

Development System Environment

Intellec MDS Series III with 64 KBytes.

Target System Environment

Any iAPX 286 system with 8274 (non-vector mode) serial link, 8254 timer, and 8259A interrupt controller such as the iSBC 286/10 Single Board Computer.

PROMs are supplied for locations 0FF8000H through 0FFFFFFH.

ORDERING INFORMATION

Part Number Description

SDM 286 iSBC 286 and iAPX 286 System Debug Monitor package including cables, PROMs, software, and operator manual

(Not available stand alone until Q3'83)

A software license must be or have been executed

Currently available only with the iSBC 286/10 ES Kit or with the iRMX 286R Kit.

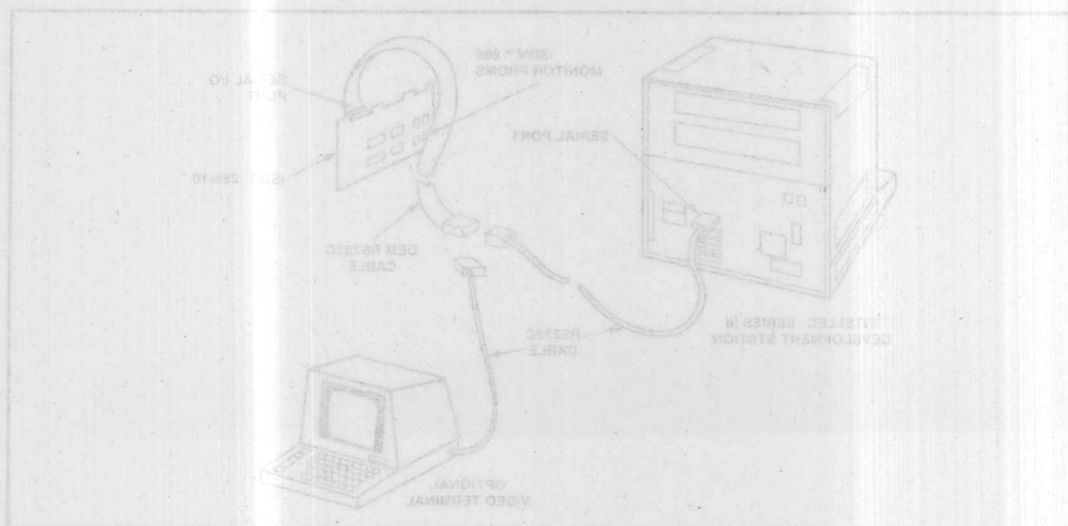


Figure 1. Typical iSDM™ 286 Environment

iRMX™ 88

REAL-TIME MULTITASKING EXECUTIVE

- Event-driven multitasking executive software supports iSBC™ 86/05, 86/12A, 88/25, 88/40 or iAPX 86, 88 based applications
- Small, high-performance, PROMable executive supports high sample rates
- Provides simple, intertask communications and synchronization
- Supports the 8087 Numeric Processor Extension (NPX) for arithmetic applications
- Supports component or iSBC™-based system generation through Interactive Configuration Utility
- I/O system provides compatible iRMX™ 86 files and device independent I/O interface
- I/O system supports the User Run-time Interface (URI) for PL/M, PASCAL and FORTRAN coded application tasks
- Memory management of full megabyte iAPX 86, 88 memory

The iRMX 88 Real-Time Multitasking Executive is a small, event-driven single-user executive system. Designed for dedicated computer applications using iSBC 86/05, 86/12A, 88/25, 88/40 or iAPX 86, 88 custom products, the modular software package provides real-time application support for PASCAL, FORTRAN, PL/M, and assembler coded tasks. Application tasks utilize intertask communications, asynchronous I/O control, priority-based resource allocation and file support for the iSBC 204, 208, 215, 215/218, and 220 Disk Controllers, and the iSBC 254 Bubble Memory product.

The small, high performance iRMX 88 Executive can be located in EPROM or bootstrapped into RAM memory. The iRMX 88 Executive offers features that are suitable for performance-critical process control applications, production test stand units, sophisticated laboratory analysis, instrumentation, specialized data acquisition systems or monitoring stations. The iRMX 88 design, based upon the iRMX 80 Real-Time Executive, offers iRMX 80-like interfaces for those 8-bit applications which are upgrading to 16-bit solutions for the 1 Megabyte addressing, expanded application functions, and higher performance data sampling requirements.

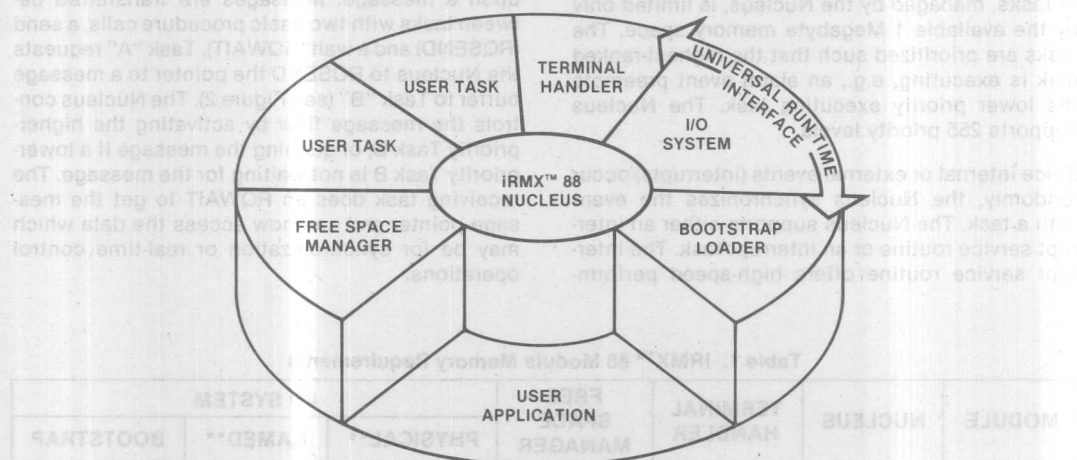


Figure 1. Module Representation

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

FUNCTIONAL DESCRIPTION

The iRMX 88 Real-Time Multitasking Executive Software package provides facilities for executing tasks concurrently, managing resources and servicing asynchronous events to users of Intel's single board computers and custom iAPX 86, 88-based products. The foundation modules support real-time dedicated computer applications with priority-based task scheduling, interrupt dispatching, real-time clock control with 1 ms resolution, multiple event monitoring and control, and file services for flexible, hard, Winchester, SMD disk units and bubble memory devices. The software package includes the primary modules: Nucleus, Free Space Manager, Terminal Handler, I/O System and Bootstrap Loader. The Interactive Configuration Utility (ICU) executes on a Series III Inteltec System, or iRMX 86 Operating System with a Universal Development Interface (UDI).

FEATURE OVERVIEW

Event-Driven Multitasking

The iRMX 88 Executive provides a control software foundation called a Nucleus. The iRMX 88 Nucleus provides two major functions: first, the facility for concurrent task execution; secondly, the facility for handling simultaneous asynchronous events.

The structured multitasking environment permits segmenting of the application tasks. The number of tasks, managed by the Nucleus, is limited only by the available 1 Megabyte memory space. The tasks are prioritized such that the highest-ranked task is executing, e.g., an alarm event preempts the lower priority executing task. The Nucleus supports 255 priority levels.

Since internal or external events (interrupts) occur randomly, the Nucleus synchronizes the event with a task. The Nucleus supports either an interrupt service routine or an interrupt task. The interrupt service routine offers high-speed perform-

ance flexibility since it masks all interrupts and supports burst-rate data sample gathering. The interrupt task is useful for lower frequency interrupts, masking only lower priority interrupts.

Small High-Performance Executive

The iRMX 88 Executive software utilizes a simple, straightforward architecture which minimizes the memory requirements, as shown in Table 1. In addition, the modules are designed to be totally EPROM resident for those systems where mass storage devices cannot be used because of the danger of contamination.

Real-time microcomputer solutions require the recognition of interrupts. The performance of the system is with respect to data sample rates. If there is no activity in progress when an interrupt occurs, the time to handle that interrupt is dependent on the number of instructions executed, e.g., 175 microseconds interrupt latency time on an iSBC 86/12A board. Most real-time solutions have multiple events occurring and background operations in progress. Seldom does a background task have critical sections of code which cannot be interrupted.

Intertask Communications

The iRMX 88 Nucleus provides a simple, easy-to-use intertask communications mechanism based upon a message. Messages are transferred between tasks with two basic procedure calls, a send (RQSEND) and a wait (RQWAIT). Task "A" requests the Nucleus to RQSEND the pointer to a message buffer to Task "B" (see Figure 2). The Nucleus controls the message flow by activating the higher-priority Task B, or queuing the message if a lower-priority Task B is not waiting for the message. The receiving task does an RQWAIT to get the message pointer and can now access the data which may be for synchronization or real-time control operations.

Table 1. iRMX™ 88 Module Memory Requirements

MODULE	NUCLEUS	TERMINAL HANDLER	FREE SPACE MANAGER	I/O SYSTEM		
				PHYSICAL**	NAMED**	BOOTSTRAP
EPROM* (K bytes)	3.0	1.3	0.6	20.0	32.0	0.5

* amount of code configured in EPROM; all numbers are approximate

** includes one 3K byte device driver (named file plus physical file is 34.0K bytes)

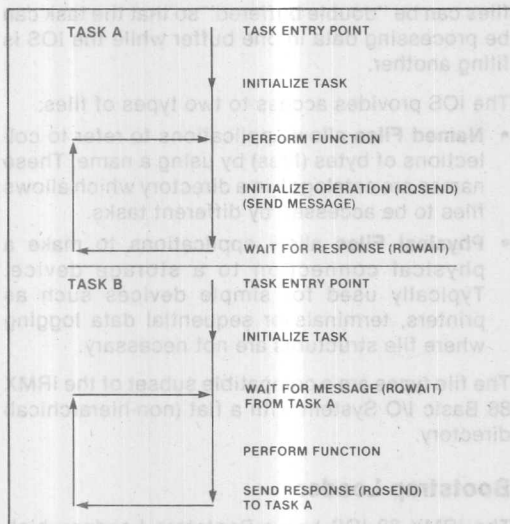


Figure 2. Intertask Communications

Numeric Data Processor

The iRMX 88 Nucleus fully supports the 8087 Numeric Processor Extension (NPX) functions for high-speed arithmetic functions of real-time applications. High-performance numeric processing applications, which utilize 8-, 16-, 32- and 64-bit integers, 32-, 64- and 80-bit floating point or 18-digit BCD operations, are accelerated up to 100 times over a iAPX 86, 88 software solution. The NPX functions, including trigonometric, logarithmic and exponential functionals, are essential in scientific, engineering, navigational or military applications.

Nucleus Primitives

The Nucleus performs other functions as shown in Table 2, in addition to the message communications management. Some primitives like CREATE TASK and DELETE TASK allow dynamic creation/deletion of tasks during run-time. This dynamic capability allows the Nucleus tables to

Table 2. Nucleus Primitives

NAME	FUNCTION
ACCEPT	Accept a message from specified exchange. Returns message address if available, zero otherwise.
CREATE TASK	Create task by building new Task Descriptor based on specified Static Task Descriptor.
CREATE EXCHANGE	Create exchange at specified RAM address.
DISABLE INTERRUPT	Disable specified interrupt level.
DELETE EXCHANGE	Delete specified exchange.
DELETE TASK	Delete the task specified.
ENABLE INTERRUPT	Initialize message portion of the Interrupt Exchange Descriptor associated with the specified interrupt level (the first time called only), and enable specified interrupt level.
END INTERRUPT	Signals specific end-of-interrupt for the specified interrupt exchange in a user-supplied interrupt service routine.
INTERRUPT SEND	Send an interrupt message to the specified interrupt exchange.
RESUME	Resume a task that has previously been suspended.
SEND	Send the message located at "msg-addr" to the exchange specified by "exch-addr."
SET INTERRUPT	Set interrupt vector address. An interrupt is to be serviced by the user-supplied routine starting at the address, thus bypassing Nucleus interrupt software.
SUSPEND TASK	Suspend execution of the task specified by the Task Descriptor.
WAIT	Wait at the specified exchange until a message is available or time limit expires. Return address of system timeout message or user message.

expand and accommodate infrequently used tasks which are loaded into memory from a mass storage device.

Interactive System Generation

The iRMX 88 Executive is constructed in a thoroughly modular manner with the full range of facilities being offered in library modules. By selecting the appropriate features and combining them with the user-written application tasks the generated system is tailored to the application's requirements minimizing memory overhead for unused features.

An Interactive Configuration Utility provides a query-based tool that configures the iRMX 88-based application. Responding to questions from the ICU utility program executing on a Series III Intellec Microcomputer Development System or an iRMX 86-based system, the user quickly tailors the real-time application system.

I/O System

The iRMX 88 I/O System provides an extensive facility for device-independent I/O. Through a series of supplied iRMX 86 compatible device drivers, the I/O System supports a wide-range of iSBC peripheral controllers. Custom peripheral controllers are supported through user-written device drivers which are integrated with the I/O System at system configuration time. The device-independent nature of the system allows use of different devices without application redesign.

The I/O System (IOS) procedures manage real-time file operations supporting both sequential and random access (see Table 3). The IOS maximizes system throughput by allowing multiple disk operations to proceed in parallel. For example,

files can be "double buffered" so that the task can be processing data in one buffer while the IOS is filing another.

The IOS provides access to two types of files:

- **Named Files** allow applications to refer to collections of bytes (files) by using a name. These names are cataloged in a directory which allows files to be accessed by different tasks.
- **Physical Files** allow applications to make a physical connection to a storage device. Typically used for simple devices such as printers, terminals or sequential data logging where file structures are not necessary.

The file types are a compatible subset of the iRMX 86 Basic I/O System with a flat (non-hierarchical) directory.

Bootstrap Loader

The iRMX 88 IOS has a Bootstrap Loader which loads a file from mass storage into system memory. The configurable Bootstrap Loader loads the file from a specific device, automatically from the first-ready device of a designated device list, or accepts the file name from a terminal. Storing the system software on disk allows easier future changes to the application system.

Run-Time Interface

The iRMX 88 Executive provides the User Run-time Interface (URI). This URI interface, in addition to encompassing the I/O System services, provides additional functionality for tasks. The additional functionality includes a trap function and memory management routines which provide the run-time foundation for PASCAL-86, FORTRAN-86, or PL/M-86 coded application tasks.

Table 3. I/O System Services

	SERVICE	FUNCTION
Data Transfer Services	CLOSE OPEN READ SEEK TRUNCATE WRITE	Closes a file connection. Opens a file connection for access. Reads a number of bytes from a file. Seeks to the indicated position. Truncates a file. Writes a number of bytes to that file.
File Connection Services	ATTACH CREATE CONNECTION STATUS DELETE DETACH RENAME	Attaches to a file connection. Creates a file and returns a file connection. Returns the file connection status . Marks the file for deletion . Detaches a file connection. Renames an existing file.
Volume Preparation	FORMAT	Formats the disk for files.

SPECIFICATIONS

Intellec® System Configuration and Generation Requirements

Series III Intellec Microcomputer Development System with UDI support and a minimum of 2 diskette drives.

IRMX™-Based Configuration and Generation Requirements

IRMX 86-based system with UDI support and a minimum of 2 diskette drives.

Supported Hardware

ISBC™ SUPPORTED MICROCOMPUTERS

ISBC 86/05 Board

ISBC 86/12A Board

ISBC 88/25 Board

ISBC 88/40 Board

MASS STORAGE

ISBC 204 Flexible Diskette Controller

ISBC 208 Flexible Disk Controller

ISBC 215A Winchester Disk Controller

ISBC 215B Winchester Disk Controller

ISBC 220 SMD Disk Controller

ISBC 254 Bubble Memory Board

MULTIMODULE™ BOARDS

iSBX 218 Flexible Disk Controller (when used with the iSBC 215 Controller)

iSBC 337 Numeric Data Processor

iSBX 351 Serial I/O Board

CUSTOM IAPX 86, 88-BASED SYSTEMS REQUIREMENTS

8253 or 8254 Programmable Interval Timer

8259A Programmable Interrupt Controller

8251A USART or iSBX 351 board (when the Terminal Handler is configured into the system).

8087 Numeric Processor Extension (when NPX tasks are configured into the system).

Reference Manuals (supplied)

143238 — Introduction to the IRMX 80/88 Real-Time Multitasking Executives

143241 — IRMX 88 Installation Instructions

143232 — IRMX 88 Reference Manual

142603 — IRMX 80/88 Interactive Configuration User's Guide

142926 — Guide to Writing Device Drivers for the IRMX 86 and IRMX 88 I/O Systems

ORDERING INFORMATION

Part Number Description

RMX 88	A licensed product which includes Nucleus, Terminal Handler, Free Space Manager, and I/O System object modules. Package also includes UDI-compatible Interactive Configuration Utility program for system generation and a complete set of manuals. Purchase price includes an iRMX 88 Customer Training Course credit.
RMX 88 ARO	Single Density ISIS media. Requires derivative work incorporation fee.
RMX 88 BRO	Double Density ISIS media. Requires derivative work incorporation fee.
RMX 88 DRO	Single Density RMX-86 media. Requires derivative work incorporation fee.

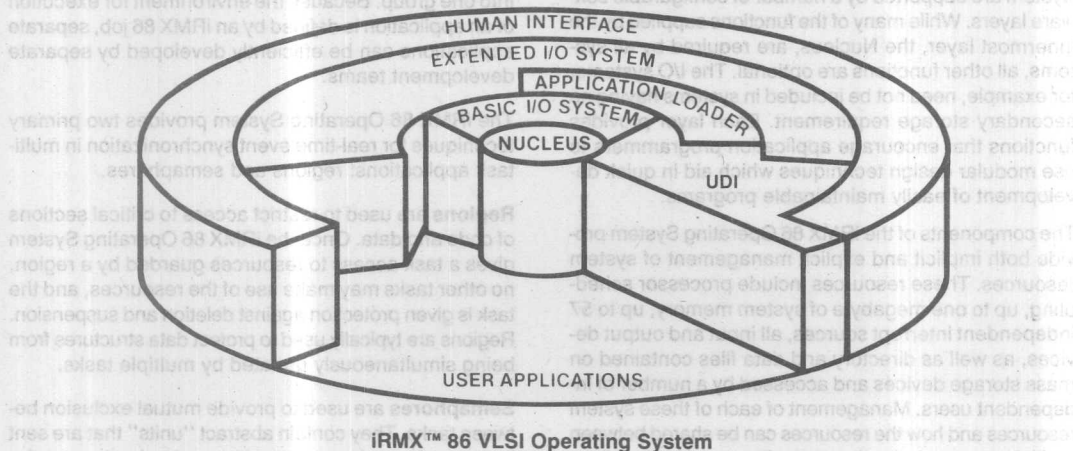
Part Number Description

RMX 88 ABY	Single Density ISIS media. Includes incorporation fee buyout.
RMX 88 BBY	Double Density ISIS media. Includes incorporation fee buyout.
RMX 88 DBY	Single Density RMX-86 media. Includes incorporation fee buyout.
RMX 88 AWX	One year Single Density ISIS media update service.
RMX 88 BWX	One year Double Density ISIS media update service.
RMX 88 DWX	One year Single Density RMX-86 media update service.
RMX 88 LST	Human readable source listings for iRMX 88 software.
RMX 88 LWX	Update service for human readable source listings.
RMX 88 RF	Incorporation fee.

iRMX™ 86 OPERATING SYSTEM

- Real-time processor management for time-critical iAPX 86 and iAPX 88 applications
- On-target system development with Universal Development Interface (UDI)
- Configurable system size and function for diverse application requirements
- All iRMX™ 86 code can be (P)ROM'ed to support totally solid state designs
- Compatible operating system services for iAPX 86/30 and 88/30 Operating System Processors (iOSP™ 86)
- Multi-terminal support with multi-user human interface
- Broad range of device drivers included for industry standard MULTIBUS® peripheral controllers
- Expandable to multi-processor systems with iMMX™ 800 Message Exchange Software
- Extendable to iAPX 286 systems with iRMX™ 286R option
- Powerful utilities for interactive configuration and real-time debugging

The iRMX™ 86 Operating System is an easy-to-use, real-time, multi-tasking and multi-programming software system designed to manage and extend the resources of iSBC® 86 and iSBC 88 Single Board Computers, as well as other iAPX 86- and iAPX 88-based microcomputers. iRMX 86 functions are available in silicon with the iAPX 86/30 and 88/30 Operating System Processors, in a user configurable software package, and fully integrated into the SYSTEM 86/300 Family of Microcomputer Systems. The Operating System provides a number of standard interfaces that allow iRMX 86 applications to take advantage of industry standard device controllers, hardware components, and a multitude of software packages developed by Independent Software Vendors (ISVs). Many high-performance features extend the utility of iRMX 86 Systems into applications such as data collection, transaction processing, and process control where immediate access to advances in VLSI technology is paramount. These systems may deliver real-time performance and explicit control over resources; yet also support applications with multiple users needing to simultaneously access terminals. The configurable layers of the System provide services ranging from interrupt management and standard device drivers for many sophisticated controllers, to data-file maintenance commands provided by a comprehensive multi-user human interface. By providing access to the standard Universal Development Interface (UDI) for each user terminal, Original Equipment Manufacturers (OEMs) can pass program development and target application customization capabilities to their users.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iOSP, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

The iRMX 86 Operating System is a complete set of system software modules that provide the resource management functions needed by computer systems. These management functions allow Original Equipment Manufacturers (OEMs) to best use resources available in microcomputer systems while getting their products to market quickly, saving time and money. Engineers are relieved of writing complex system software and can concentrate instead on their application software.

This data sheet describes the major features of the iRMX 86 Operating System. The benefits provided to engineers who write application software and to users who want to take advantage of improving microcomputer price and performance are explained. The first section outlines the system resource management functions of the Operating System and describes several system calls. The second section gives a detailed overview of iRMX 86 features aimed at serving both the iRMX 86 system designer and programmer, as well as the end users of the product into which the Operating System is incorporated.

FUNCTIONAL DESCRIPTION

To take best advantage of iAPX 86 and 88 microprocessors in applications where the computer is required to perform many functions simultaneously, the iRMX 86 Operating System provides a multiprogramming environment in which many independent, multi-tasking application programs may run. The flexibility of independent environments allows application programmers to separately manage each application's resources during both the development and test phases.

The resource management functions of the iRMX 86 System are supported by a number of configurable software layers. While many of the functions supplied by the innermost layer, the Nucleus, are required by all systems, all other functions are optional. The I/O systems, for example, need not be included in systems having no secondary storage requirement. Each layer provides functions that encourage application programmers to use modular design techniques which aid in quick development of easily maintainable programs.

The components of the iRMX 86 Operating System provide both implicit and explicit management of system resources. These resources include processor scheduling, up to one megabyte of system memory, up to 57 independent interrupt sources, all input and output devices, as well as directory and data files contained on mass storage devices and accessed by a number of independent users. Management of each of these system resources and how the resources can be shared between multiple processors and users is discussed in the following sections.

Process Management

To implement multi-tasking application systems, programmers require a method of managing the different processes of their application, and for allowing the processes to communicate with each other. The Nucleus layer of the iRMX 86 System provides a number of facilities to efficiently manage these processes, and to effectively communicate between them. These facilities are provided by system calls that manipulate data structures called tasks, jobs, semaphores, regions, and mailboxes. The iRMX 86 System refers to these structures as "objects".

Tasks are the basic element of all applications built on the iRMX 86 Operating System. Each task is an entity capable of executing CPU instructions and issuing system calls in order to perform a function. Tasks are characterized by their register values (including those of an optional 8087 Numeric Processor Extension), a priority between 0 and 255, and the resources associated with them.

Each iRMX 86 task in the system is scheduled for operation by the iRMX 86 nucleus. Figure 1 shows the five states in which each task may be placed, and some examples of how a task may move from one state to another. The iRMX 86 nucleus ensures that each task is placed in the correct state, defined by the events in its external environment and by the task issuing system calls. Each task has a priority to indicate its relative importance and need to respond to its environment. The nucleus guarantees that the highest priority ready-to-run task is the task that runs.

Jobs are used to define the operating environment of a group of tasks. Jobs effectively limit the scope of an application by collecting all of its tasks and other objects into one group. Because the environment for execution of an application is defined by an iRMX 86 job, separate applications can be efficiently developed by separate development teams.

The iRMX 86 Operating System provides two primary techniques for real-time event synchronization in multi-task applications: regions and semaphores.

Regions are used to restrict access to critical sections of code and data. Once the iRMX 86 Operating System gives a task access to resources guarded by a region, no other tasks may make use of the resources, and the task is given protection against deletion and suspension. Regions are typically used to protect data structures from being simultaneously updated by multiple tasks.

Semaphores are used to provide mutual exclusion between tasks. They contain abstract "units" that are sent between the tasks, and can be used to implement the cooperative sharing of resources.

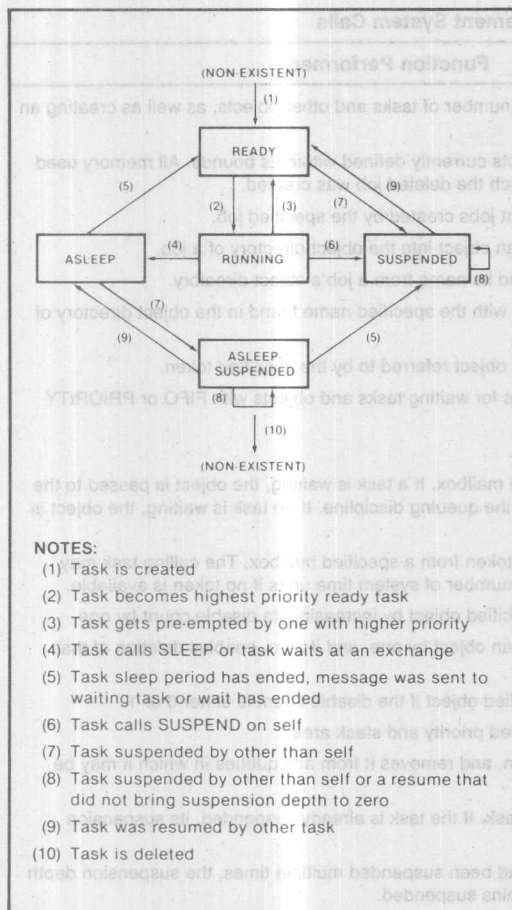


Figure 1. Task State Diagram

Multi-tasking applications must communicate information and share system resources among cooperating tasks. The iRMX 86 Operating System assigns a unique 16-bit number, called a token, to each object created in the System. Any task in possession of this token is able to access the object. The iRMX 86 Nucleus allows tasks to gain access to objects, and hence system resources, at run-time with two additional mechanisms: mailboxes and object directories.

Mailboxes are used by tasks wishing to share objects with other tasks. A task may share an object by sending the object's token via a mailbox. The receiving task can check to see if a token is there, or can wait at the mailbox until a token is present.

Object Directories are also used to make an object available to other tasks. An object is made public by cataloging its token and name in a directory. In this manner,

any task can gain access to the object by knowing its name, and job environment that contains the directory.

Three example jobs are shown in Figure 2 to demonstrate how two tasks can share an object that was not known to the programmers at the time the tasks were developed. Both Job 'A' and Job 'B' exist within the environment of the 'Root Job' that forms the foundation of all iRMX 86 systems. Each job possesses a directory in which tasks may catalog the name of an object. Semaphore 'RS', for example, is accessible by all tasks in the system, because its name is cataloged in the directory of the Root Job. Mailbox 'AN' can be used to transfer objects between Tasks 'A2' and 'A3' because its token is accessible in the object directory for Job 'A'.

Table 1 lists the major functions of the iRMX 86 Nucleus that manage system processes.

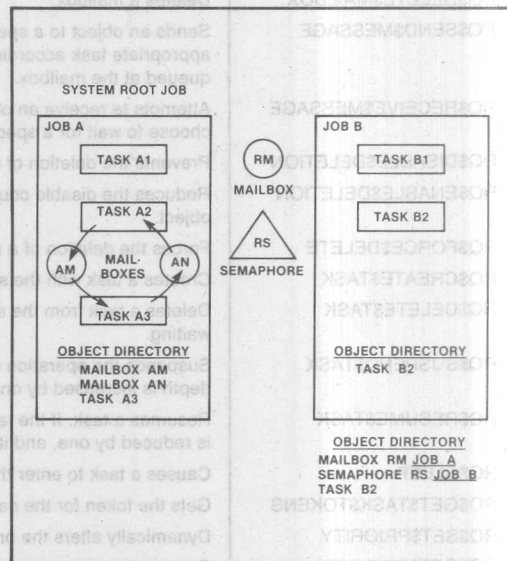


Figure 2. Object Directories

Memory Management

Each job in an iRMX 86 System defines the amount of the one megabyte of addressable memory to be used by its tasks. The iRMX 86 Operating System manages system memory and allows jobs to share this critical resource by providing another object type: segments.

Segments are contiguous pieces of memory, between 16 Bytes and 64K Bytes in length, that exist within the environment of the job in which they were created. Segments form the fundamental piece of system memory used for task stacks, data storage, system buffers, loading programs from secondary storage, passing information between tasks, etc.

Table 1. Process Management System Calls

System Call	Function Performed
RQ\$CREATE\$JOB	Creates an environment for a number of tasks and other objects, as well as creating an initial task and its stack.
RQ\$DELETE\$JOB	Deletes a job and all the objects currently defined within its bounds. All memory used is returned to the job from which the deleted job was created.
RQ\$OFFSPRING	Provides a list of all the current jobs created by the specified job.
RQ\$CATALOG\$OBJECT	Enters a name and token for an object into the object directory of a job.
RQ\$UNCATALOG\$OBJECT	Removes an object's token and its name from a job's object directory.
RQ\$LOOKUP\$OBJECT	Returns a token for the object with the specified name found in the object directory of the specified job.
RQ\$GET\$TYPE	Returns a code for the type of object referred to by the specified token.
RQ\$CREATE\$MAILBOX	Creates a mailbox with queues for waiting tasks and objects with FIFO or PRIORITY discipline.
RQ\$DELETE\$MAILBOX	Deletes a mailbox.
RQ\$SEND\$MESSAGE	Sends an object to a specified mailbox. If a task is waiting, the object is passed to the appropriate task according to the queuing discipline. If no task is waiting, the object is queued at the mailbox.
RQ\$RECEIVE\$MESSAGE	Attempts to receive an object token from a specified mailbox. The calling task may choose to wait for a specified number of system time units if no token is available.
RQ\$DISABLE\$DELETION	Prevents the deletion of a specified object by increasing its disable count by one.
RQ\$ENABLE\$DELETION	Reduces the disable count of an object by one, and if zero, enables deletion of that object.
RQ\$FORCE\$DELETE	Forces the deletion of a specified object if the disable count is either 0 or 1.
RQ\$CREATE\$TASK	Creates a task with the specified priority and stack area.
RQ\$DELETE\$TASK	Deletes a task from the system, and removes it from any queues in which it may be waiting.
RQ\$SUSPEND\$TASK	Suspends the operation of a task. If the task is already suspended, its suspension depth is increased by one.
RQ\$RESUME\$TASK	Resumes a task. If the task had been suspended multiple times, the suspension depth is reduced by one, and it remains suspended.
RQ\$SLEEP	Causes a task to enter the ASLEEP state for a specified number of system time units.
RQ\$GET\$TASK\$TOKENS	Gets the token for the calling task or associated objects within its environment.
RQ\$SET\$PRIORITY	Dynamically alters the priority of the specified task.
RQ\$GET\$PRIORITY	Obtains the current priority of a specified task.
RQ\$CREATE\$REGION	Creates a region, with an associated queue of FIFO or PRIORITY ordering discipline.
RQ\$DELETE\$REGION	Deletes the specified region if it is not currently in use.
RQ\$ACCEPT\$CONTROL	Gains control of a region only if the region is immediately available.
RQ\$RECEIVE\$CONTROL	Gains control of a region. The calling task may specify the number of system time units it wishes to wait if the region is not immediately available.
RQ\$SEND\$CONTROL	Relinquishes control of a region.
RQ\$CREATE\$SEMAPHORE	Creates a semaphore.
RQ\$DELETE\$SEMAPHORE	Deletes a semaphore.
RQ\$SEND\$UNITS	Increases a semaphore counter by the specified number of units.
RQ\$RECEIVE\$UNITS	Attempts to gain a specified number of units from a semaphore. If the units are not immediately available, the calling task may choose to wait.

The example in Figure 2 also demonstrates when information is shared between Tasks 'A2' and 'A3'; 'A2' only needs to create a segment, put the information in the memory allocated, and send it via the Mailbox 'AM' using the RQ\$SEND\$MESSAGE system call (see Table 1). Task 'A3' would get the message by using the RQ\$RECEIVE\$MESSAGE system call. The Figure also shows how the receiving task could signal the sending task by sending an acknowledgement via the second Mailbox 'AN'.

Each job is created with both maximum and minimum limits set for its memory pool. Memory required by all objects and resources created in the job is taken from this pool. If more memory is required, a job may be allowed to borrow memory from the pool of its containing job (the job from which it was created). In this manner, initial jobs may efficiently allocate memory to jobs they subsequently create, without exactly knowing their requirements.

The iRMX 86 Operating System supplies other memory management functions to search specific address ranges for available memory. The System performs this search at system initialization, and can be configured to ignore non-existent memory and addresses reserved for I/O devices and other application requirements.

Table 2 lists the major system calls used to manage the system memory.

Interrupt Management

Real-time systems, by their nature, must respond to asynchronous and unpredictable events quickly. The iRMX 86 Operating System uses interrupts and the event-driven nucleus described earlier to give real-time response to events. Use of a pre-emptive scheduling technique ensures that the servicing high priority events always take precedence over other system activities.

The iRMX 86 Operating System gives applications the flexibility to optimize either interrupt response time or interrupt response capability by providing two tiers of Interrupt Management. These two distinct tiers are managed by Interrupt Handlers and Interrupt Tasks.

Interrupt Handlers are the first tier of interrupt service. For small, simple functions, interrupt handlers are often the most efficient means of responding to an event. They provide faster response than interrupt tasks, but must be kept simple since interrupts (except the iAPX 86 and 88 non-maskable interrupt) are masked during their execution. When extended interrupt service is required, interrupt handlers "signal" a waiting interrupt task that, in turn, performs more complicated functions.

Interrupt Tasks are distinct tasks whose priority is associated with a hardware interrupt level. They are permitted

to make any iRMX 86 system call. While an interrupt task is servicing an interrupt, interrupts of lower priority are not allowed to pre-empt the system.

Table 3 shows the iRMX 86 System Calls provided to manage interrupts.

INTERRUPT MANAGEMENT EXAMPLE

Figure 3 illustrates how the iRMX 86 Interrupt System may be used to output strings of characters to a printer. In the example, a mailbox named 'PRINT' is used by all tasks in the system to queue messages to be printed. Application tasks put the characters in segments that are transmitted to the printer interrupt service task via the PRINT Mailbox. Once printing is complete, the same interrupt task passes the messages on to another application via the FINISHED Mailbox so that an operator message can be displayed.

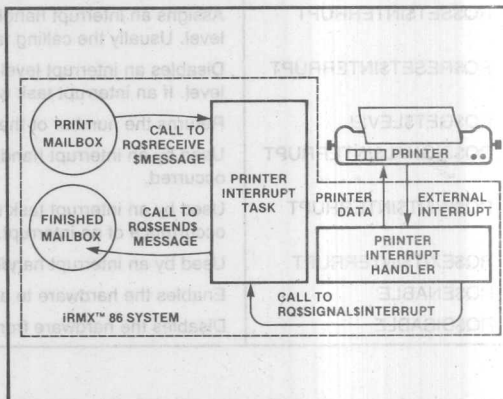


Figure 3. Interrupt Management Example

BASIC I/O SYSTEM

The Basic I/O System (BIOS) provides the direct access to I/O devices needed by real-time applications. The BIOS allows I/O functions to overlap other system functions. In this manner, application tasks make asynchronous calls to the iRMX 86 BIOS, and proceed to perform other activities. When the I/O request must be completed before an application can continue, the task waits at a mailbox for the result of the operation.

Some system calls provided by the BIOS are listed in Table 4.

The Basic I/O System communicates with peripheral devices through device drivers. These device drivers provide the System with four basic functions needed to control and communicate with devices: Initialize I/O, Finish I/O, Queue I/O, and Cancel I/O. Using the device driver interface, users of non-standard devices may write custom drivers compatible with the I/O System.

Table 2. Memory Management System Calls

System Call	Function Performed
RQ\$CREATE\$SEGMENT	Dynamically allocates a memory segment of the specified size.
RQ\$DELETE\$SEGMENT	Deletes the specified segment by deallocating the memory.
RQ\$GET\$POOL\$ATTRIBUTES	Returns attributes such as the minimum and maximum, as well as current size of the memory in the environment of the calling task's job.
RQ\$GET\$SIZE	Returns the size (in bytes) of a segment.
RQ\$SET\$POOL\$MIN	Dynamically changes the minimum memory requirements of the job environment containing the calling task.

Table 3. Interrupt Management System Calls

System Call	Function Performed
RQ\$SET\$INTERRUPT	Assigns an interrupt handler and, if desired, an interrupt task to the specified interrupt level. Usually the calling task becomes the interrupt task.
RQ\$RESET\$INTERRUPT	Disables an interrupt level, and cancels the assignment of the interrupt handler for that level. If an interrupt task was assigned, it is deleted.
RQ\$GET\$LEVEL	Returns the number of the highest priority interrupt level currently being processed.
RQ\$SIGNAL\$INTERRUPT	Used by an interrupt handler to signal the associated interrupt task that an interrupt has occurred.
RQ\$WAIT\$INTERRUPT	Used by an interrupt task to SLEEP until the associated interrupt handler signals the occurrence of an interrupt.
RQ\$EXIT\$INTERRUPT	Used by an interrupt handler to relinquish control of the System.
RQ\$ENABLE	Enables the hardware to accept interrupts from a specified level.
RQ\$DISABLE	Disables the hardware from accepting interrupts at or below a specified level.

Table 4. Key BIOS I/O Management System Calls

System Call	Function Performed
RQ\$A\$ATTACH\$FILE	Creates a Connection to an existing file.
RQ\$A\$CHANGE\$ACCESS	Changes the types of accesses permitted to the specified user(s) for a specific file.
RQ\$A\$CLOSE	Closes the Connection to the specified file so that it may be used again, or so that the type of access may be changed.
RQ\$A\$CREATE\$DIRECTORY	Creates a Named File used to store the names and locations of other Named Files.
RQ\$A\$CREATE\$FILE	Creates a data file with the specified access rights.
RQ\$A\$DELETE\$CONNECTION	Deletes the Connection to the specified file.
RQ\$A\$GET\$FILE\$STATUS	Returns the current status of a specified file.
RQ\$A\$OPEN	Opens a file for either read, write, or update access.
RQ\$A\$READ	Reads a number of bytes from the current position in a specified file.
RQ\$A\$SEEK	Moves the current data pointer of a Named or Physical file.
RQ\$A\$WRITE	Writes a number of bytes at the current position in a file.
RQ\$WAIT\$I/O	Synchronizes a task with the I/O System by causing it to wait for I/O operation results.

The iRMX 86 Operating System includes a number of device drivers to allow applications to use standard USART serial communication devices, multiple CRTs and keyboards, bubble memories, diskettes, disks, a Centronics-type parallel printer, and many of Intel's iSBC and iSBX™ device controllers (see Table 9). If an application requires use of a non-standard device, users need only write a device driver to be included with the BIOS, and access it as if it were part of the standard system. For most random-access devices, this job is further simplified by using standard routines provided with the System. Use of this technique ensures that applications can remain device independent.

MULTI-TERMINAL SUPPORT

The iRMX 86 Terminal Support provides line editing and terminal control capabilities. The Terminal Support communicates with devices through simple drivers that do only character I/O functions. Dynamic terminal reconfiguration is provided so that attributes such as terminal type and line speed may be changed without modifying the application or the Operating System. Dynamic configuration may be typed in, generated programmatically or stored in a file and copied to a terminal I/O connection.

The iRMX 86 Terminal Support provides automatic translation of control characters to specific control sequences for each terminal. This translation enables applications using standard control characters to function with non-standard terminals. The translation requirements for each terminal can be stored in terminal description files and copied to a connection, as described above.

DISK I/O PERFORMANCE

Table 5 shows iRMX 86 performance obtained using the iSBC 215 Winchester Disk and iSBX 218 Diskette Controllers under the specified conditions.

Each device driver can be used to interface to a number of separate and, in some cases, different devices (See Figure 4). The iSBC 215 Device Driver, supplied with the system, is capable of supporting the iSBC 215 Winchester Disk Controller, the iSBC 220 SMD Disk Controller, and the iSBX 218 Flexible Disk Controller (when mounted on an iSBC 215 board). Each device controller may, in turn, control a number of separate device units. In addition, each driver may control a number of like device controllers. This capability allows the use of large storage systems with a minimum of I/O system code to write or maintain.

EXTENDED I/O SYSTEM

The iRMX 86 Extended I/O System (EIOS) adds a number of I/O management capabilities to simplify access to

files. Whereas the BIOS provides users with the basic system calls needed for direct management of I/O resources, many users prefer to have the system perform all the buffering and synchronization of I/O requests automatically. The EIOS allows users to access I/O devices without having to write procedures for buffering data, or to specify particular devices with constant device names.

By performing device buffering automatically, the iRMX 86 EIOS optimizes accesses to disks and other devices. Often, when an application task asks the System to READ a portion of a file, the System is able to respond immediately with the data it has read in advance of the request. Similarly, the EIOS will not delay a task for writing data to a device unless it is specifically told to, or if its output buffers are filled.

Logical file and device names are provided by the EIOS to give applications complete file and device independence. Applications may send data to the 'line printer' (:LP:) without needing to know which specific device will be used as the printer. This logical name may, in fact, not be a printer at all, but it could be a disk file that is later scheduled for printing.

The EIOS uses the functions provided by the BIOS to synchronize individual I/O requests with results returned by device drivers. Most EIOS system calls are similar to the BIOS calls, except that they appear to suspend the operation of the calling task until the I/O requests are completed.

Table 5. BIOS Typical Performance

Function	Average Character Throughput Bytes per Second*	
	Winchester Disk	Diskette
Single File Read	42,000	15,800
Two File Read (Same Device)	36,800	5,700
Single File Write	23,800	5,400
Two File Write (Different Devices)	36,200	6,900
Read/Write Two Files (Different Devices)	38,900	6,000

* These measurements were made in the following environment: Entire iRMX™ 86 operating system and application code and data located in on-board RAM of a 8-MHz iSBC® 86/30 Single Board Computer. Named files, each with a file size of 128 KBytes, were used with a device and volume granularity of 1 KBytes and six 1 KByte buffers. The disk interleave factor was 2. The iSBC 215 Winchester Controller was attached to two 20-Mbyte drives, and supported the iSBX™ 218 Diskette Controller that, in turn, was attached to two double density 8" diskette drives. This performance is, to a large part, restricted by the mechanical speed of the devices.

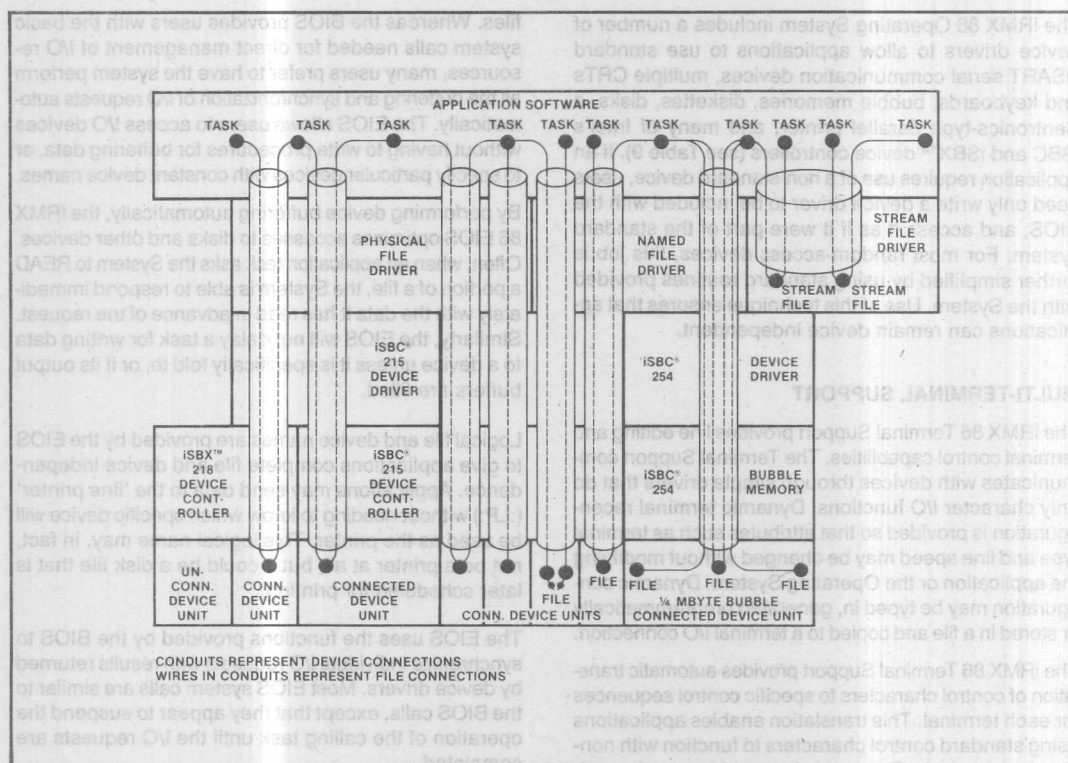


Figure 4. Device Driver and Controller Relationships

File Management

The iRMX 86 Operating System provides three distinct types of files to ensure efficient management of both program and data files: Named Files, Physical Files, and Stream Files. Each file type provides access to I/O devices through the standard device drivers mentioned earlier. The same device driver is used to access physical and named files for a given device.

NAMED FILES

Named files allow users to access information on secondary storage by referring to a file with its ASCII name. The names of files stored on a device are stored in special files called directories. As directories are themselves named files, the iRMX 86 File System allows directories to contain the names of other directories. Figure 5 illustrates the resulting hierarchical file structure. This structure is useful for isolating file names to particular user applications, and for tailoring system data to the requirements of users and applications sharing storage devices. Using different branches on the directory tree, different users do not have to coordinate in naming their files to ensure unique names.

Whenever a request is made involving a file name, the System will search the appropriate directory in order to find the necessary information about the file's size, access rights, and specific location on the storage device.

The iRMX 86 BIOS uses an efficient format for writing the directory and data information into secondary storage. This standard iRMX 86 format is fully compatible with the ISO Media standard, and other Intel systems such as the iRMX 88 Operating System. This structure enables the system to directly access any byte in a file, often without having to do additional I/O to access space allocation information. The maximum size of an individual file is 2^{32} (4.3 billion) bytes.

EASE OF ACCESS

The hierarchical file structure is provided to isolate and organize collections of named files. To give operators fast and simple access to any level within the file tree, an ATTACHFILE command is provided. This command allows operators to give a logical name to a point in the tree so that a long sequence of characters need not be typed each time a file is referred to.

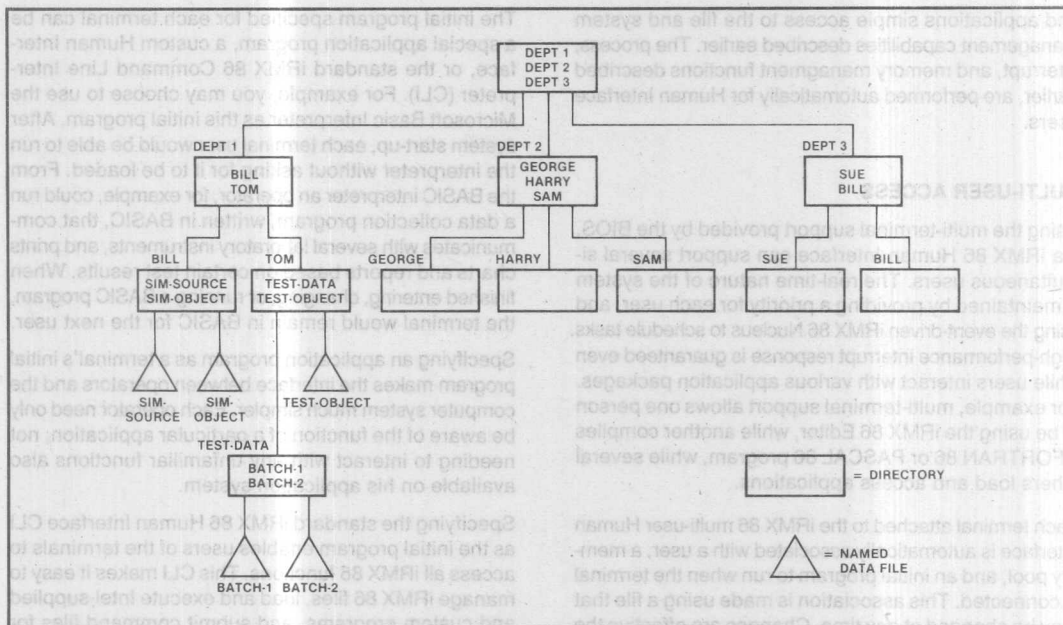


Figure 5. Hierarchical Named File Structure

ACCESS PROTECTION

Access to each Named File is protected by the rights assigned to each user by the owner of the file. Rights to read, append, update, and delete may be selectively granted to other users of the system. In general, users of Named Files are classified into one of three categories: User, Group, and World. Users and Groups are used when different programmers and programs need to share information stored in a file. The World classification is used when rights are to be granted to all who can use the system.

PHYSICAL FILES

Physical Files allow more direct device access than Named Files. Each Physical File occupies an entire device, treated as a single stream of individually accessible bytes. No access control is provided for Physical Files as they are typically used for such applications as driving a printing device, translating from one device format to another, driving a paper tape device, and controlling analog mechanisms.

STREAM FILES

Stream Files provide applications with a method of using iRMX 86 file management methods for data that does not need to go into secondary storage. Stream Files act as direct channels, through system memory, from one task to another. These channels are very useful to pro-

grams, for example, wishing to preserve file and device independence allowing data sent to a printer one time, to a disk file another time, and to another program on a different occasion.

BOOTSTRAP AND APPLICATION LOADERS

Two utilities are supplied with the System to load programs and data into system memory from secondary storage devices:

The **iRMX 86 Bootstrap Loader** can be configured to a size of less than 600 bytes of P(ROM), and is typically used to load the initial system from the system disk into memory, and begin its execution.

The **Application Loader** is typically used by application programs already running in the system to load additional programs and data from any secondary storage device. The Human Interface layer, for example, uses the Application Loader to load the non-resident Human Interface Commands. The Application Loader is capable of loading both relocatable and absolute code, as well as program overlays.

Human Interface

The flexibility of the interface between computer controlled machines and their users often determines the usability and ultimate success of the machines. Table 12 lists iRMX 86 Human Interface functions giving users

and applications simple access to the file and system management capabilities described earlier. The process, interrupt, and memory management functions described earlier, are performed automatically for Human Interface users.

MULTI-USER ACCESS

Using the multi-terminal support provided by the BIOS, the iRMX 86 Human Interface can support several simultaneous users. The real-time nature of the system is maintained by providing a priority for each user, and using the event-driven iRMX 86 Nucleus to schedule tasks. High-performance interrupt response is guaranteed even while users interact with various application packages. For example, multi-terminal support allows one person to be using the iRMX 86 Editor, while another compiles a FORTRAN 86 or PASCAL 86 program, while several others load and access applications.

Each terminal attached to the iRMX 86 multi-user Human Interface is automatically associated with a user, a memory pool, and an initial program to run when the terminal is connected. This association is made using a file that may be changed at any time. Changes are effective the next time the system is initialized.

The initial program specified for each terminal can be a special application program, a custom Human Interface, or the standard iRMX 86 Command Line Interpreter (CLI). For example, you may choose to use the Microsoft Basic Interpreter as this initial program. After system start-up, each terminal user would be able to run the interpreter without asking for it to be loaded. From the BASIC interpreter an operator, for example, could run a data collection program, written in BASIC, that communicates with several laboratory instruments, and prints charts and reports based on certain test results. When finished entering, changing, or running a BASIC program, the terminal would remain in BASIC for the next user.

Specifying an application program as a terminal's initial program makes the interface between operators and the computer system much simpler. Each operator need only be aware of the function of a particular application; not needing to interact with any unfamiliar functions also available on his application system.

Specifying the standard iRMX 86 Human Interface CLI as the initial program enables users of the terminals to access all iRMX 86 functions. This CLI makes it easy to manage iRMX 86 files, load and execute Intel-supplied and custom programs, and submit command files for later execution.

Table 6. iRMX™ Real-Time Performance

Real-Time Function	Execution Time (msec)
SUSPEND TASK	0.45
INTERRUPT LATENCY (to Handler)	0.20 (Max)
INTERRUPT LATENCY (to Handler)	0.03 (Typical)
CONTEXT SWITCH CAUSED BY INTERRUPT	0.68 (Max)
SEND MESSAGE (no context switch)	0.30
SEND MESSAGE (with context switch)	0.57
SEND CONTROL (no context switch)	0.19
SEND CONTROL (with context switching)	0.50
RECEIVE CONTROL (no waiting)	0.25

Context switch time is the time between executing in the context of a task, and the first instruction to execute in the context of another task.

These times were measured using an 8 MHz iSBC® 86/30 Single Board Computer with the standard configuration supported by the Preconfigured System, and all program and data stored in on-board dynamic RAM.

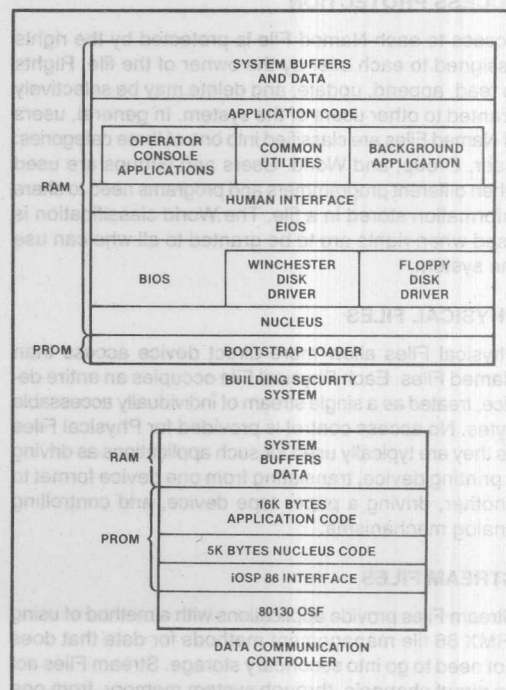
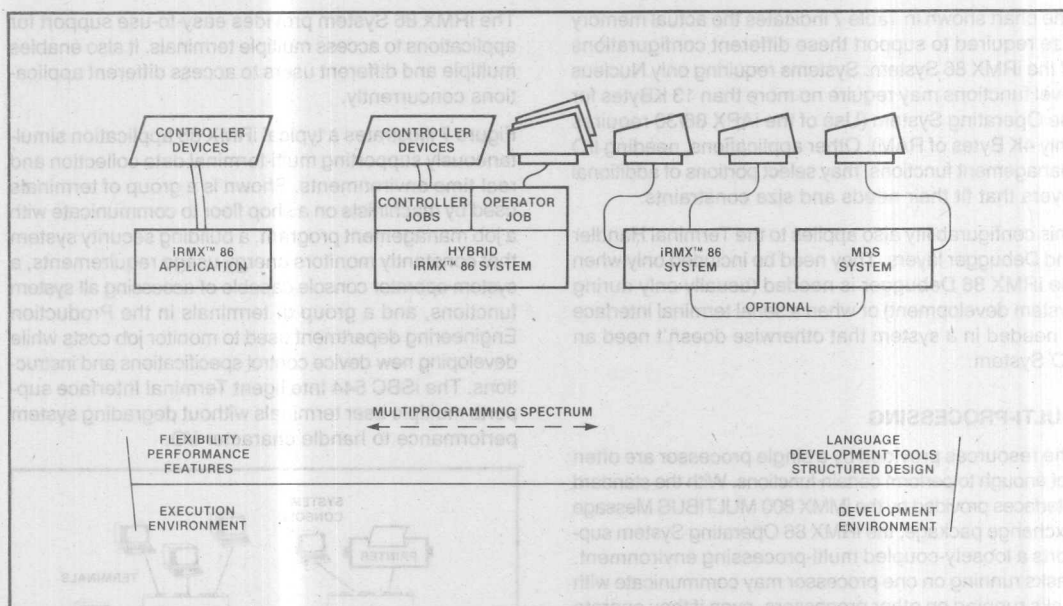


Figure 6. Typical iRMX™ 86 Configurations



FEATURE OVERVIEW

The iRMX 86 Operating System is well suited to serve the demanding needs of real-time applications executing on complex microprocessor systems. The iRMX 86 System also provides many tools and features needed by real-time system developers and programmers. The following sections describe features useful in both the development and execution environments. The description of each feature outlines the advantages given to hardware and software engineers concerned with overall system cost, expandability with custom and industry standard options, and long-term maintenance of iRMX 86-based systems. The development environment features also describe the ease with which the iRMX 86 Operating System can be incorporated into overall system designs.

Execution Environment Features

REAL-TIME PERFORMANCE

The iRMX 86 Operating System is designed to offer the high performance, multi-tasking functions required by real-time systems. Designers can make use of the latest VLSI devices such as the 8087 Numeric Processor Extension, and the 80130 Operating System Firmware Component to improve their system cost/performance ratio, or the iRMX™ 800 MULTIBUS® Message Exchange

software package to divide and coordinate various system activities among multiple processors. Typical iRMX 86 system performance characteristics are shown in Table 6.

Many real-time systems require high-performance operation. To meet this requirement, all of iRMX 86 (except for the Human Interface commands) can be put into zero wait-state P(ROM). This approach eliminates the possibility of disk access times slowing down performance, while allowing system designers to take advantage of high performance memory devices.

CONFIGURABILITY

The iRMX 86 Operating System is configurable by system layer, and by system call within each layer. In addition, all the I/O port addresses used by the System are configurable by the user. This flexibility gives designers the freedom to choose configurations of hardware and software that best suit their size and functional requirements. Two example configurations are shown in Figure 6.

Most configuration options are selected during system design stages. Others may be selected during system operation. For example, the amount of memory devoted to queues within a Mailbox can be specified at the time the Mailbox is created. Devoting more memory to the Mailbox allows more messages to be transmitted to other tasks without having to degrade system performance to allocate additional memory dynamically.

The chart shown in Table 7 indicates the actual memory size required to support these different configurations of the iRMX 86 System. Systems requiring only Nucleus level functions may require no more than 13 KBytes for the Operating System (Use of the iAPX 86/30 requires only 4K Bytes of RAM). Other applications, needing I/O management functions, may select portions of additional layers that fit their needs and size constraints.

This configurability also applies to the Terminal Handler and Debugger layers. They need be included only when the iRMX 86 Debugger is needed (usually only during system development) or when a serial terminal interface is needed in a system that otherwise doesn't need an I/O System.

MULTI-PROCESSING

The resources provided by a single processor are often not enough to perform certain functions. With the standard interfaces provided by the iMMX 800 MULTIBUS Message Exchange package, the iRMX 86 Operating System supports a loosely-coupled multi-processing environment. Tasks running on one processor may communicate with tasks running on other processors, even if they operate under different operating systems. The iMMX 800 software is capable of sending messages over the MULTIBUS to tasks operating under either the iRMX 80 8-bit Multi-Tasking Executive, the iRMX 88 Executive, or the iRMX 86 Operating System. Using this message exchange mechanism, applications may increase their system performance quite easily, improve overall interrupt response, gain access to the iSBC 550 Ethernet Controller, and leave room for future product enhancements.

MULTI-USER ACCESS

Many real-time systems must provide a variety of users access to system control functions and collected data.

The iRMX 86 System provides easy-to-use support for applications to access multiple terminals. It also enables multiple and different users to access different applications concurrently.

Figure 7 illustrates a typical iRMX 86 application simultaneously supporting multi-terminal data collection and real-time environments. Shown is a group of terminals used by machinists on a shop floor to communicate with a job management program, a building security system that constantly monitors energy usage requirements, a system operator console capable of accessing all system functions, and a group of terminals in the Production Engineering department used to monitor job costs while developing new device control specifications and instructions. The iSBC 544 Intelligent Terminal Interface supports multiple user terminals without degrading system performance to handle character I/O.

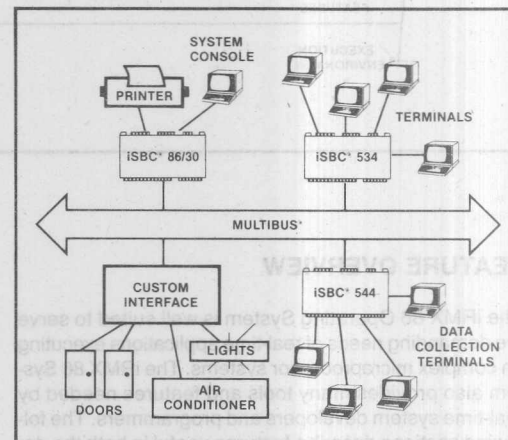


Figure 7. Multi-Terminal and Multi-User Real-Time System

Table 7. iRMX™ 86 Configuration Size Chart

System Layer	Min. ROMable Size	Max. Size	Data Size
Bootstrap Loader	0.5K	1.5K	6K*
Nucleus	10.5K	24K	2K
BIOS	26K	78K	1K
Application Loader	4K	10K	2K
EIOS	10.5K	12.5K	1K
Human Interface	22K	22K	15K
UDI	11K	11K	0
Terminal Handler	3K	3K	0.3K
Debugger	28.5K	28.5K	1K
Human Interface Commands			116K
Interactive Configuration Utility			308K

* Usable by System after bootloading.

EXTENDABILITY

The iRMX 86 Operating System provides three means of extensions. This extendability is essential for support of OEM and volume end user value added features. This ability is provided by: user-defined operating system calls, user-defined objects (similar to Jobs, Tasks, etc.), and the ability to add functions later in the product life cycle. The modular, layered structure of the System easily facilitates later additions to iRMX 86 applications. User-defined objects are supported by the functions listed in Table 8.

Using standard iRMX 86 system calls users may define custom objects, enabling applications to easily manipulate commonly used structures as if they were part of the original operating system.

EXCEPTION HANDLING

The System includes predefined exception handlers for typical I/O and parameter error conditions. The error handling mechanism is both configurable and extendable.

SUPPORT OF STANDARDS

The iRMX 86 Operating System supports the many hardware and software standards needed by most application systems to ensure that commonly available hardware and software packages may be interfaced with a minimum of cost and effort. The iRMX 86 System supports the iSBC family of products built on the Intel MULTIBUS (IEEE Standard 796), and a number of standard software interfaces such as the UDI and the common device driver interface (See Figure 8). The procedural interfaces of

Table 8. User Extension System Calls

System Call	Function Performed
RQ\$CREATE\$COMPOSITE	Creates a custom object built of previously defined objects.
RQ\$DELETE\$COMPOSITE	Deletes the custom object, but not the various objects from which it was built.
RQ\$INSPECT\$COMPOSITE	Returns a list of Token Identifiers for the component objects from which the specified composite object is built.
RQ\$ALTER\$COMPOSITE	Replaces a component object of a composite object.
RQ\$CREATE\$EXTENSION	Creates a new type of object and assigns a mailbox used for collecting these objects when they are deleted.
RQ\$DELETE\$EXTENSION	Deletes an extension definition.

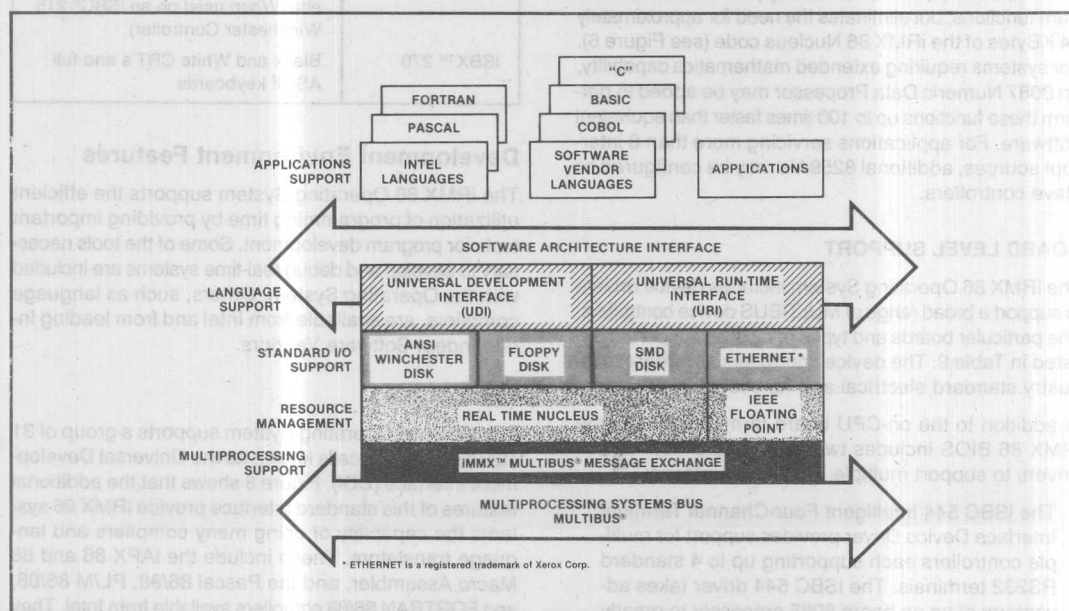


Figure 8. iRMX™ 86 Standard Interfaces

the UDI, a software analogy to the MULTIBUS, are listed in Table 10.

The Operating System includes support for the proposed IEEE 80-bit extended real-variable format of the 8087 Numeric Data Processor, the IEEE 796 (MULTIBUS) hardware interface, and the Intel Universal Run-time Interface (URI). Other standards such as the iMMX 800 MULTIBUS Message Exchange, and an Ethernet* communication interface, are supported by optional software packages available to run on the iRMX 86 System.

SPECTRUM OF CPU PERFORMANCE

The iRMX 86 System supports 8086 and 8088 based systems directly at a variety of processor clock speeds. With the iRMX 286R Operating System option, completely compatible systems can be built around the iAPX 286 processor. By choosing the appropriate CPU, designers can select from a wide range of performance without having to change application software.

COMPONENT LEVEL SUPPORT

The iRMX 86 System may be tailored to support specific hardware configurations. In addition to system memory, only an iAPX 86 or iAPX 88 microprocessor, an 8259A Programmable Interrupt Controller, and either an 8253 or 8254 Programmable Interval Timer are required. In addition, the iRMX 86 Operating System may be used to augment the functions of the 80130 Operating System Firmware Component that not only provides these hardware functions, but eliminates the need for approximately 14 KBytes of the iRMX 86 Nucleus code (see Figure 6). For systems requiring extended mathematics capability, an 8087 Numeric Data Processor may be added to perform these functions up to 100 times faster than equivalent software. For applications servicing more than 8 interrupt sources, additional 8259A's may be configured as slave controllers.

BOARD LEVEL SUPPORT

The iRMX 86 Operating System includes device drivers to support a broad range of MULTIBUS device controllers. The particular boards and types of devices supported are listed in Table 9. The device controllers all adhere to industry standard electrical and functional interfaces.

In addition to the on-CPU board terminal drivers, the iRMX 86 BIOS includes two iSBC board-level device drivers to support multiple terminal interfaces:

The iSBC 544 Intelligent Four-Channel Terminal Interface Device Driver provides support for multiple controllers each supporting up to 4 standard RS232 terminals. The iSBC 544 driver takes advantage of an on-board 8085 processor to greatly reduce the system processor time required for ter-

minal I/O by locally managing input and output buffers. The iSBC 544 firmware provided with the operating system can off-load the system CPU by as much as 75%.

The iSBC 534 Four-Channel USART Controller Device Driver also provides support for multiple controller boards each supporting up to 4 standard RS232 terminals.

Table 9. Supported Devices

iSBC® Device Controller	Supported Devices
iSBC® 86,88	Serial Port to CRT, Parallel Port to Centronics-type Printer, Interval Timer, and Interrupt Controller
iSBC® 204	Single Density Diskette
iSBC® 206	Cartridge-type Hard Disk
iSBC® 208	Single & Double Density, Single & Double Sided, 8" & 5.25" Diskette
iSBC® 215	Standard Winchester Disks
iSBC® 220	Standard Storage Module Disks
iSBC® 254	Bubble Memory Board
iSBC® 534,544	4-Channel Serial Ports to CRTs, Modems
iSBX™ 218	Single & Double Density, Single & Double Sided, 8" & 5.25" Diskette (When used on an iSBC® 215 Winchester Controller)
iSBX™ 270	Black and White CRT's and full ASCII keyboards

Development Environment Features

The iRMX 86 Operating System supports the efficient utilization of programming time by providing important tools for program development. Some of the tools necessary to develop and debug real-time systems are included with the Operating System. Others, such as language compilers, are available from Intel and from leading Independent Software Vendors.

LANGUAGES

The iRMX 86 Operating System supports a group of 31 standard system calls known as the Universal Development Interface (UDI). Figure 8 shows that the additional features of this standard interface provide iRMX 86 systems the capability of using many compilers and language translators. These include the iAPX 86 and 88 Macro Assembler, and the Pascal 86/88, PL/M 86/88, and FORTRAN 86/88 compilers available from Intel. They also include a number of other Intel development tools,

* Ethernet is a trademark of Xerox Corporation.

and language translators and utilities available from other software vendors. A subset of the UDI System Calls provides another standard interface called the Universal Runtime Interface (URI). The URI calls are those required to execute a compiled program, while the full set of UDI calls is required to run a compiler.

These standard software interfaces (the URI and the UDI) ensure that users of the iRMX 86 Operating System may transport their applications to future releases of the iRMX 86 Operating System and other Intel and independent vendor software products. The calls available in the URI and UDI are shown in Table 10.

Table 10. URI and UDI System Calls

System Call	Function Performed
Memory Management:	
DQ\$ALLOCATE	Creates a Segment of a specified size.
DQ\$FREE	Returns the specified segment to the System.
DQ\$GET\$SIZE*	Returns the size of the specified Segment.
DQ\$RESERVE\$I/O\$MEMORY*	Reserves memory to OPEN and ATTACH files.
File Management:	
DQ\$ATTACH	Creates a Connection to a specified file.
DQ\$CHANGES\$ACCESS*	Changes the user access rights associated with a file or directory.
DQ\$CHANGES\$EXTENSION	Changes the extension of a file name in memory.
DQ\$CLOSE	Closes the specified file Connection.
DQ\$CREATE	Creates a Named File.
DQ\$DELETE	Deletes a Named File.
DQ\$DETACH	Closes a Named File and deletes its Connection.
DQ\$OPEN	Opens a file for a particular type of access.
DQ\$GET\$CONNECTION\$STATUS*	Returns the current status of the specified file Connection.
DQ\$FILE\$INFO*	Returns data about a file Connection.
DQ\$READ	Reads the next sequence of bytes from a file.
DQ\$RENAME*	Renames the specified Named File.
DQ\$SEEK	Moves the position pointer of a file.
DQ\$TRUNCATE	Truncates a file.
DQ\$WRITE	Writes a sequence of bytes to a file.
Process Management:	
DQ\$EXIT	Exits from the current application job.
DQ\$OVERLAY*	Causes the specified overlay to be loaded.
DQ\$SPECIAL	Performs special I/O related functions on terminals with special control features.
DQ\$TRAP\$CC	Captures control when CNTRL/C is typed.
Exception Handling:	
DQ\$GET\$EXCEPTION\$HANDLER	Returns a pointer to the program currently being used to process errors.
DQ\$DECODE\$EXCEPTION	Returns a short description of the specified error code.
DQ\$TRAP\$EXCEPTION	Identifies a custom exception processing program for a particular type of error.
Application Assistance:	
DQ\$DECODE\$TIME	Returns system time and date in binary and ASCII character format.
DQ\$GET\$ARGUMENT*	Returns the next argument from the character string used to invoke the application program.
DQ\$GET\$SYSTEM\$ID*	Returns the name of the underlying operating system supporting the UDI.
DQ\$GET\$TIME*	Returns the current time of day as kept by the underlying operating system.
DQ\$SWITCH\$BUFFER	Selects a new buffer from which to process commands.

* Calls available only through the UDI.

The high performance of the iRMX 86 Operating System enhances the throughput of compilers and other development utilities. Table 11 indicates the average performance of typical development environment functions operating in the same configuration described in Table 5.

Table 11. Development Environment Performance

Function	Average Execution Time
Directory Command (S Format with 25 files)	5.3 sec
Load the COPY Command (Winchester to Winchester)	1.2 sec
Copy a 1K Byte File (Winchester to Winchester)	1.0 sec
Copy a 16K Byte File	1.7 sec
Copy a 64K Byte File	3.9 sec
Copy a 1K Byte File (Winchester to Diskette)	1.4 sec
Compile PL/M 86	393 lpm
Compile PASCAL 86 Program	453 lpm

TOOLS

Certain tools are necessary for the development of microcomputer applications. The iRMX 86 Human Interface includes many of these tools as non-resident commands. They can be included on the system disk of an application system, and brought into memory when needed to perform functions as listed in Table 12.

Table 12. Major Human Interface Utilities

Command	Function
BACKUP	Copy directories and files from one device to another.
COPY	Copy one or more files to one or more destination files.
CREATEDIR	Create a directory file to store the names of other files.
DIR	List the names, sizes, owners, etc. of the files contained in a directory.
ATTACHFILE	Give a logical name to a specified location in a file directory tree.
PERMIT	Grant or rescind user access to a file.
RENAME	Change the name of a file.
SUBMIT	Start the processing of a series of commands stored in a file.
SUPER	Change operator's ID to that of the System Manager with global access rights and privileges.

Table 12. Major Human Interface Utilities (Con't.)

Command	Function
TIME	Set the system time-of-day clock.
VERIFY	Verify the structure of an iRMX™ 86 Named File volume, and check for possible disk data errors.

INTERACTIVE CONFIGURATION UTILITY

The iRMX 86 Operating System is designed to provide OEMs the ability to configure for specific system hardware and software requirements. The Interactive Configuration Utility (ICU) builds iRMX 86 configurations by asking appropriate questions and making reasonable assumptions. It runs on either an Intellec® Series III Development System or iRMX 86 System supporting the UDI and a hard disk. Table 13 lists the hardware and support software requirements of different iRMX 86 development system environments.

Table 13. iRMX™ 86 Development Environment

Intellec® Series III: MDS 313 PL/M 86/88 Compiler One hard disk and one diskette drive
iRMX™ 86 Preconfigured System iRMX™ 860 Utility iRMX™ 863 PL/M 86/88 Compiler iSBC® 957B Monitor 448K Bytes of RAM 5M Byte On-Line Storage and one double-density diskette drive
SYSTEM 86/330 Microcomputer System Basic configuration

Figure 9 shows one of the many screens displayed during the process of defining a configuration. It shows the abbreviations for each choice on the left, a more complete description with the range of possible answers in the center, and the current (sometimes default) choice on the right. The bottom of the screen shows three changes made by the operator (lower case lettering), and a request for help on the Exception Mode question. In response to a request for help, the ICU displays an additional screen outlining possible choices and some overall system effects.

The ICU requests only information required as a result of previous choices. For example, if no Extended I/O System functions are required, the ICU will not ask any further questions about the EIOS. Once a configuration session is complete, the operator may save all the information in a file. Later, when small changes are neces-

sary, this file can be modified. A completely new session is not required.

Nucleus		
(ASC)	All Sys Calls [Yes/No]	Yes
(PV)	Parameter Validation [Yes/No]	Yes
(ROD)	Root Object Directory Size [0 - 0FFFF]	0014H
(MTS)	Minimum Transfer Size [0 - 0FFFF]	0040H
(DEH)	Default Exception Handler [Yes/No/Deb/Use]	Yes
(NEH)	Name of Ex Handler Object Module [1 - 32chs]	
(EM)	Exception Mode [Never/Program/Environ/All]	Never
(NR)	Nucleus in ROM [Yes/No]	No

Enter Changes [Abbreviations ? = new-value] : ASC = N
 :pv = no
 :rod = 48
 :em ?

Figure 9. ICU Screen for iRMX™ 86 Nucleus

REAL-TIME DEBUGGING TOOLS

The iRMX 86 Operating System supports three distinct debugging environments: Static, Dynamic, and Post-Mortem. While the iRMX 86 Operating System does support a multi-user Human Interface, these real-time debugging aids are usually most useful in a single-user environment where modifications made to the system cannot affect other users.

The static debugging aid is the iSBC 957B Monitor (included in the first shipment of some iRMX 86 options). The Monitor provides a basic debugging capability for both system and application code. The iRMX 86 Debugger provides a dynamic system debugging tool for testing and debugging real-time systems. The Debugger allows programmers to stop and inspect one task while the rest of the system continues to operate. The iRMX 86 Crash/Dump Analyzer enables programmers to inspect a system's structure after a problem has caused it to stop normal operation. Each of these debugging facilities are described below.

iSBC® 957B Monitor

The iSBC 957B Monitor can be used either as a stand-alone monitor for static debugging and system start-up, or as a communication link to an Inteltec Development System. A number of PROMs are included along with the necessary cables to control a hardware configuration such as is pictured in Figure 10. All programs necessary for the Inteltec system and the target system are included. Configuration tools for users wishing to support different hardware configurations are also included.

Debugging of any iAPX 86 or 88 application is accomplished in an interactive manner via either of the two terminals shown in Figure 10. If an Inteltec Development System is not present, all debugging instructions necessary to view and modify register and memory contents, set execution breakpoints and provide single instruction

execution are accessible from a terminal connected directly to the iAPX 86 or 88 system.

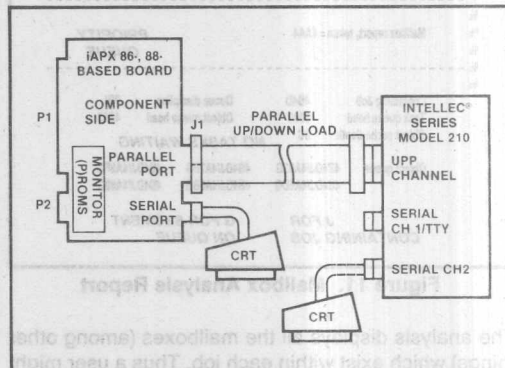


Figure 10. Typical iSBC® 957B Configuration

iRMX™ 86 Debugger

The iRMX 86 Debugger runs as part of an iRMX 86 application. It may be used at any time during program development, or may be integrated into an OEM system to aid in the discovery of latent errors. The Debugger can be used to search for errors in any task, even while the other tasks in the system are running. The iRMX 86 Debugger communicates with the developer via a terminal handler that supports full line editing.

System Crash/Dump Analyzer

The often difficult job of debugging real-time applications is made much simpler with the System Crash/Dump Analyzer. The analyzer allows program developers to record system memory for later analysis even if the system has halted. This analysis lists such vital information as which jobs have active tasks, which system queues contain which tasks, and what segments contain which data.

The information used by the Analyzer is obtained from a copy of iRMX 86 data structures after a fault has caused an unexpected halt (crash). The processor also may be halted deliberately to perform a system analysis. The system information is created by a two-step process:

1. Transferring an image of iRMX 86 system memory to a disk file on an Inteltec Series III Microcomputer Development system.
2. Later printing an analyzed and formatted printout description of the state of the system.

Figure 11 shows a portion of a Crash/Dump Analyzer display for an iRMX 86 Mailbox. The display identifies the mailbox by its token and shows its key attributes. This information is followed by a list of tokens for objects (if any) queued at the mailbox.

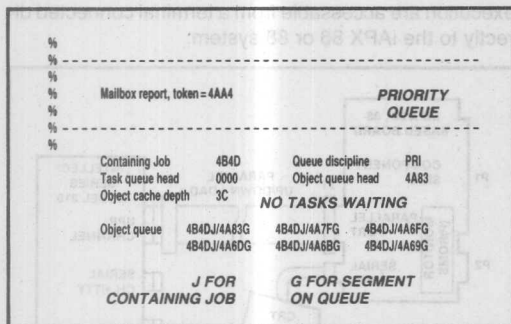


Figure 11. Mailbox Analysis Report

The analysis displays all the mailboxes (among other things) which exist within each job. Thus a user might learn critical information by observing a number of objects different from that expected.

Performance problems can be identified under some circumstances. Noticing that certain mailboxes frequently have many objects queued may suggest an increase in the high performance cache size for the mailbox to improve its throughput, or give the designer cause to investigate the receiving task operation to see why the queue is so large.

The analyzer automatically checks for system inconsistencies such as corrupted data structures, incorrect object types, and stack overflow. Reports of such problems accompany the reports on specific system objects.

PARAMETER VALIDATION

Some iRMX 86 System Calls require parameters that may change during the course of developing iRMX 86 applications. The iRMX 86 Operating System includes an optional set of routines to validate these parameters to ensure that correct numeric values are used, and that correct object types are used where the System expects to manipulate an object. For systems based only on the iRMX 86 Nucleus, these routines may be removed to improve the performance and code size of the System once the development phase is completed.

PRECONFIGURED SYSTEM

A ready-to-run, multi-user, Preconfigured System is included in each iRMX 86 KIT. Its configuration supports

the full complement of devices shown in Figure 12. The shaded area of the Figure represents the minimum hardware required by the start-up system. Other combinations of the devices, up to the full compliment shown, that support additional on-line storage are also possible. The Preconfigured System includes all iRMX 86 System Calls and the complete Universal Development Interface (UDI). The UDI supports Intel High-Level Languages and many applications available from Intel and many Independent Software Vendors.

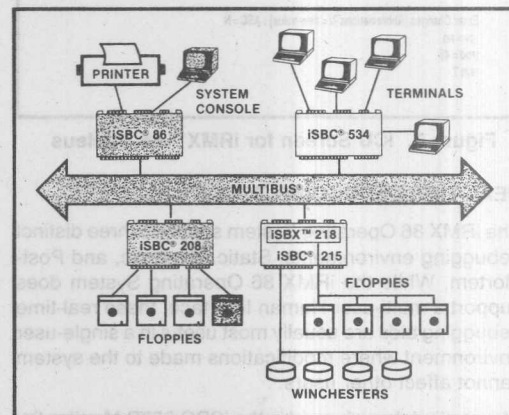


Figure 12. Pre-Configured iRMX™ 86 System

The Preconfigured System is intended to aid the initial use of iRMX 86 features. Any 8086-based system currently supporting an iRMX 86 environment with a double density diskette may simply plug in the start-up system and run. Further, this start-up system may be used to run the ICU (if a Winchester disk is attached to the system) to develop custom configurations such as those pictured in Figure 7. As shipped, the Human Interface supports a single user terminal. However, the Preconfigured System user terminal file may be altered easily to support from two to five users.

This System is also available as a separate product (order code RMX 86PC E) for those iRMX 86 users that do not require the ability to tailor their system to custom hardware and software configurations. The SYSTEM 86/300 Family of Microcomputer Systems also provide users immediate access to programming tools and system applications with a ready-to-load preconfigured iRMX 86 Operating System.

SPECIFICATIONS

Supported Software Products

iRMX 286R iRMX 86-compatible Operating System extension for iAPX 80286

iRMX 860 iRMX 86 Development Utilities Package including the iAPX 86 and 88 Linker, Locator, Macro Assembler, Librarian, and the iRMX 86 Editor
iRMX 861 PASCAL 86/88 Compiler

- iRMX 862 FORTRAN 86/88 Compiler
- iRMX 863 PL/M 86/88 Compiler
- iRMX 864 TX Screen-oriented Editor
- iMMX 800 MULTIBUS Message Exchange software package for iRMX 80, 86, and 88 application systems
- iOSP 86 Support Package for iAPX 86/30 and 88/30 Operating System Processors

Supported Hardware Products

COMPONENTS

- iAPX 86 and 88 Microprocessors
- iAPX 286 Microprocessors (with iRMX 286R)
- 8087 Numeric Data Processor Extension
- iAPX 86/30 (80130) Operating System Firmware Component (with iOSP 86)
- 8253 and 8254 Programmable Interval Timers
- 8259A Programmable Interrupt Controller
- 8251A USART
- 8255 Programmable Parallel Interface

iSBC® MULTIBUS® BOARD AND SYSTEM PRODUCTS

- iSBC 86/12A, 86/05, 86/14, 86/30, 88/25, and 88/40 Single Board Computers
- iSBC 286/10 Single Board Computer (With iRMX 286R)
- iSBC 204 Diskette Controller
- iSBC 206 Hard Disk Controller
- iSBC 208 Diskette Controller
- iSBC 215 Winchester Disk Controller
- iSBC 220 SMD Disk Controller
- iSBC 254 Bubble Memory System
- iSBC 534 4-Channel Terminal Interface
- iSBC 544 Intelligent 4-channel Terminal Interface and Controller
- iSBX 218 Diskette Controller (with iSBC 215)
- iSBX 350 Parallel Port (Centronics-type Printer Interface)
- iSBX 351 Serial Communications Port
- iSBX 270 CRT, Light Pen and Keyboard Interface
- SYSTEM 86/330 Computer System
- SYSTEM 86/380 Computer System

Available Literature

The iRMX 86 Documentation Set is comprised of following reference manuals. Each is also be available under the order numbers shown.

- Introduction to the iRMX 86 Operating System (9803124-04)
- iRMX 86 Operator's Manual (144523-001)
- Master Index for iRMX 86 Release 5 Documentation (145015-001)
- Getting Started With The Release 5 iRMX 86 System (145073-001)
- iRMX 86 Installation Guide (9803125-05)
- iRMX Configuration Guide (9803126-05)
- iRMX 86 Nucleus Reference Manual (9803122-04)
- iRMX 86 Terminal Handler Reference Manual (143324-002)
- iRMX 86 Debugger Reference Manual (143323-002)
- iRMX 86 Basic I/O System Reference Manual (9803123-05)
- iRMX 86 Loader Reference Manual (143318-002)
- iRMX 86 Extended I/O System Reference Manual (143308-002)
- iRMX 86 Human Interface Reference Manual (9803202-003)
- Guide to Writing Device Drivers for the iRMX 86 and iRMX 88 I/O Systems (142926-004)
- iRMX 86 Programming Techniques (142982-003)
- User's Guide For The iSBC 957B iAPX 86, 88 Interface and Execution Package (143979-002)
- iRMX 86 Disk Verification Utility Reference Manual (144133-002)
- Runtime Support Manual for iAPX 86, 88 Applications (121776-002)
- iRMX 86 Crash Analyzer Reference Manual (144522-001)

OPTIONAL REFERENCE MATERIALS

- Edit Reference Manual (143587-002)
- Guide to Using iRMX 86 Languages (142907-001)

APPLICATION NOTES

- Ap Note 86 — iRMX 86 Realtime Multitasking Operating System
- Ap Note 130 — Using Operating System Processors to Simplify Microcomputer Designs

TRAINING COURSES

- Introduction to the iRMX 86 Operating System
- Advanced iRMX 86 Operating System Concepts

CUSTOMER SEMINARS

Contact Local Intel Sales Office for details on available video-tape and slide presentations.

PRECONFIGURED iRMX™ 86 OPERATING SYSTEM

- Ready-to-run Preconfigured iRMX™ 86 Operating System for iSBC® systems
- Efficient realtime multitasking scheduler with 255 priority levels
- Complete support of 8087 numeric processor extension
- Direct support of independent software vendor compilers and applications
- Direct support for Intel on-target compilers and development tools
- Simple program load and debug with Bootstrap and Monitor in 2732A EPROMs
- Device drivers included for up to four diskettes, serial terminal interface, and parallel line printer
- A complete, high-performance, execution engine for UDI applications

The Intel Preconfigured iRMX 86 Operating System is a flexible, realtime, and multitasking system which is configured to run on a low-cost, iSBC 86-based hardware system. The iRMX 86 Operating System is designed to provide a structured and efficient environment for many time- and performance-critical applications such as factory automation, business data and text processing, medical electronics, data communications and process control. The Preconfigured System provides this environment without requiring specific hardware and software configurations. Based on the UDI software interface architecture for optional compilers and interpreters, the iRMX 86 PC System supports development of sophisticated applications using the target hardware. A ready-to-use comprehensive human interface provides advanced services including creating and maintaining a hierarchical file system, entering the debug monitor and backing-up diskette volumes.

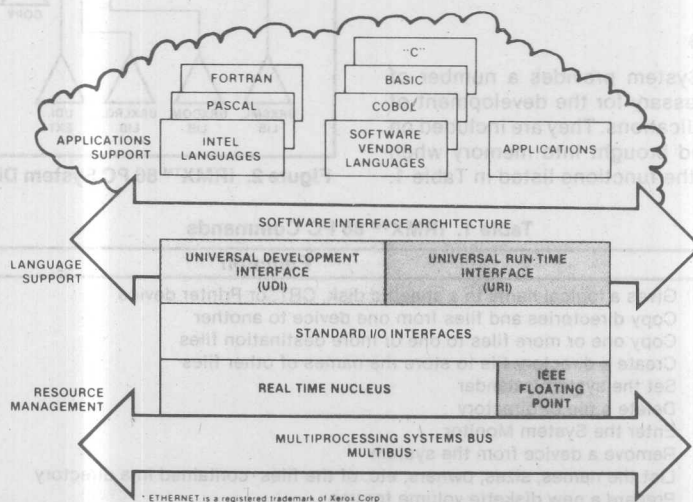


Figure 1. iRMX™ 86 PC Support for Standard Interfaces

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

The Preconfigured iRMX 86 Operating System is a complete set of system software modules that are ready-to-run in a simple MULTIBUS system consisting of an iSBC 86 computer, memory, and a diskette controller board. All the features of the iRMX 86 Operating System are provided along with a bootstrap monitor to load the system diskette into the system.

The Preconfigured iRMX 86 System provides both implicit and explicit management of system resources. These resources include the processor's time and registers, up to one megabyte of system memory, independent interrupt sources, all input and output devices, as well as directory and data files contained on up to four diskettes.

FUNCTIONAL DESCRIPTION

In applications where computers are required to perform many functions simultaneously, the iRMX 86 Operating System provides a multiprogramming environment in which many independent, and optionally multitasking, applications may run. Each application environment may be treated separately to allow application programmers the flexibility to separately manage each application's resources. A complete description of the iRMX 86 Operating System can be found in the iRMX 86 Data Sheet (Order Number: 210330).

User Commands

The iRMX 86 PC System provides a number of powerful tools necessary for the development of microcomputer applications. They are included on the system disk and brought into memory when needed to perform the functions listed in Table 1.

These commands are especially useful for managing user programs and data stored on diskettes.

File Management

The iRMX 86 PC file management system allows users to access information on diskettes by referring to a file with its ASCII name. The names of files stored on a disk are catalogued in special files called directories. As directories are themselves named files, the iRMX 86 file system allows directories to contain the names of other directories. This leads to a hierarchical file structure as illustrated in Figure 2. This structure is useful for isolating file names of particular applications, and for tailoring the system's data to the requirements of users and applications sharing storage devices.

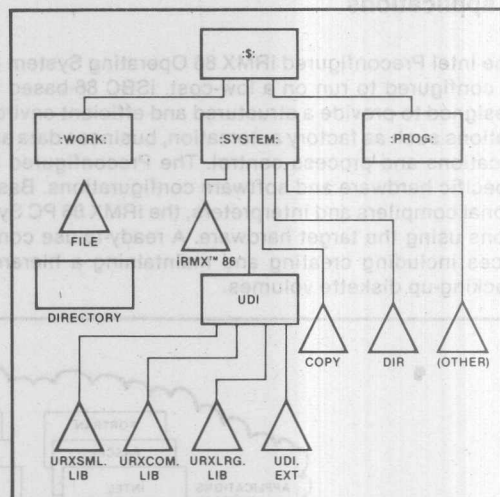


Figure 2. iRMX™ 86 PC System Disk Directory Tree

Table 1. iRMX™ 86 PC Commands

Command	Function
ATTACHDEVICE	Gives a logical name to a specific disk, CRT, or Printer device
BACKUP	Copy directories and files from one device to another
COPY	Copy one or more files to one or more destination files
CREATEDIR	Create a directory file to store the names of other files
DATE	Set the system calendar
DELETE	Delete a file or directory
DEBUG	Enter the System Monitor
DETACHDEVICE	Remove a device from the system
DIR	List the names, sizes, owners, etc. of the files contained in a directory
FORMAT	Prepare a new diskette volume for use
RENAME	Change the name of a file
RESTORE	Recreates a volume saved by BACKUP
SUBMIT	Start the processing of a series of commands stored in a file
TIME	Set the system time-of-day clock
VERIFY	Verify the structure of an iRMX 86 Named File volume, and check for possible disk data errors

Figure 2 also shows the structure of the directories on the iRMX 86 PC system diskette. It contains all the programs and commands that make up the iRMX 86 PC System. Users may add other files and directories anywhere in the structure. Whenever an operator makes a request to use one of these files, the System will search the appropriate directory tree in order to find the necessary information about the file's size, access rights, and specific location on the diskette. Applications may also refer to a specific file or group of files by specifying the directory from which to start the search.

Standard Interfaces

The iRMX 86 PC System supports a group of 25 easy-to-use standard system calls known as the

Universal Development Interface (UDI). Figure 1 shows how this interface provides iRMX 86 systems the capability of using many compilers and language translators. These include the iAPX 86 and 88 Macro Assembler, and the PASCAL 86/88, PL/M 86/88, and FORTRAN 86/88 compilers available from Intel. They also include a number of other Intel development tools, and language translators and applications available from independent software vendors.

The standard UDI software interface establishes a path to future Intel software products and opens the door to a host of compilers, interpreters, and application programs available from independent software vendors. These UDI calls are easy-to-use and are listed in Table 2. A more complete list of all the system calls provided by the iRMX 86 PC System can be found in the iRMX 86 Data Sheet.

Table 2. UDI System Calls

System Call	Function Performed
Memory Management:	
DQ\$ALLOCATE	Creates a segment of a specified size for use by the application.
DQ\$FREE	Returns the specified segment to the system.
DQ\$GET\$SIZE	Returns the size of the specified segment.
File Management:	
DQ\$ATTACH	Creates a connection to a specified file.
DQ\$CHANGE\$EXTENSION	Changes or adds an extension to a file name.
DQ\$CLOSE	Closes the specified file connection.
DQ\$CREATE	Creates a Named File for use by the application.
DQ\$DELETE	Deletes a Named File.
DQ\$DETACH	Closes a Named File and deletes its connection.
DQ\$OPEN	Opens a file for a particular type of access.
DQ\$READ	Reads the next sequence of bytes from a file.
DQ\$RENAME	Renames the specified Named File.
DQ\$SEEK	Moves the current position pointer of a file.
DQ\$TRUNCATE	Truncates a file to the specified length.
DQ\$WRITE	Writes a sequence of bytes to a file.
Process Management:	
DQ\$EXIT	Exits from the current application job.
DQ\$GET\$CONNECTION\$STATUS	Returns the current status of the specified file connection.
DQ\$OVERLAY	Causes the specified overlay to be loaded.
DQ\$SPECIAL	Performs special I/O related functions on terminals with special control features.
Exception Handling:	
DQ\$GET\$EXCEPTION\$HANDLER	Returns a pointer to the program currently being used to process errors.
DQ\$DECODE\$EXCEPTION	Returns a short description of the specified error code.
DQ\$TRAP\$EXCEPTION	Identifies a custom exception processing program for a particular type of error.

Table 2. UDI System Calls (con't.)

System Call	Function Performed
Application Assistance:	
DQ\$GET\$ARGUMENT	Returns the next argument from the character string used to invoke the application program.
DQ\$GET\$SYSTEM\$ID	Returns the name of the underlying operating system supporting the UDI.
DQ\$GET\$TIME	Returns the current time of day as kept by the underlying operating system.
DQ\$SWITCH\$BUFFER	Selects a new buffer from which to process commands.

Simple System Start-Up

The iRMX 86 PC system includes a comprehensive Monitor and Bootstrap Loader in four 2732A EPROMs. These programs have been configured to support the hardware shown in Figure 3. As shown, the Monitor is capable of communicating with an Intellec Microcomputer Development System. This communications link can be used to transfer programs and data between an iRMX 86 System and the Intellec Development System.

This start-up system provides a perfect environment for the development and efficient execution of applications programs. When these programs require different I/O devices or a different software configuration, they can be moved to any other

iRMX 86 System directly. The iRMX 86 PC System includes a separate diskette with the complete set of iRMX 86 multitasking system calls for those programmers requiring more function than is supplied by the UDI.

Debugging Aids

The iRMX 86 PC System includes a System Monitor that provides the capability of debugging one task at a time. The monitor includes instructions for examining and modifying the contents of all 8086 and 8087 registers, setting system breakpoints, single-stepping, examining and modifying system memory, executing CPU I/O, and disassembling program instructions.

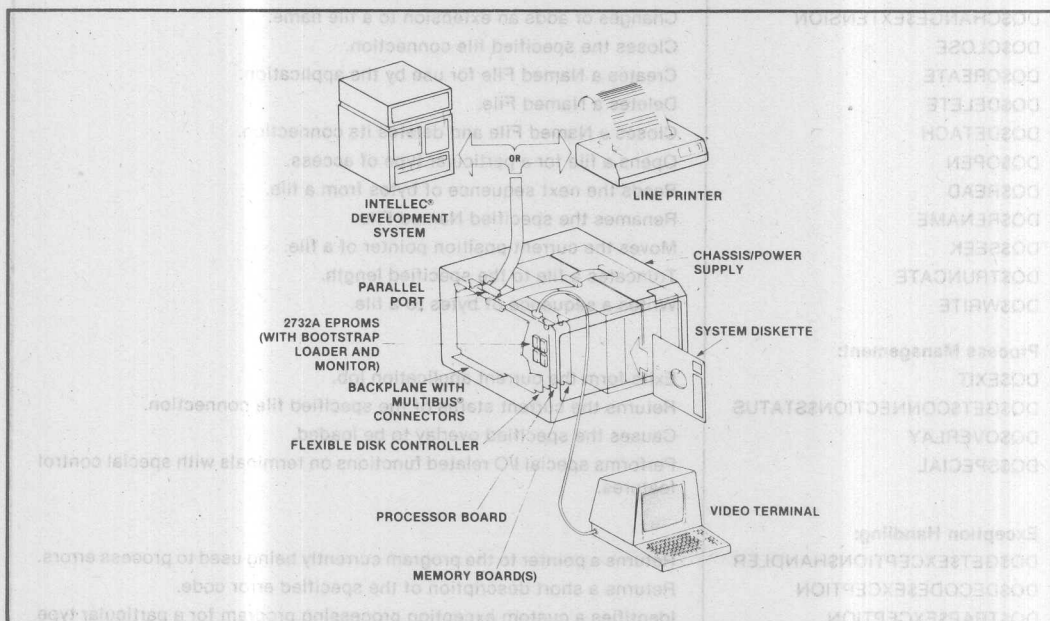


Figure 3. Hardware Configuration of PC System

SPECIFICATIONS

Optional Intel® Software Products

- iRMX 86** Fully configurable iRMX 86 Realtime Operating System
- iRMX 860** iRMX 86 Development Utilities Package including the iAPX 86 and 88 Linker, Locator, and Macro Assembler, Librarian, and the iRMX 86 Editor
- iRMX 861** PASCAL 86/88 Compiler for execution on iRMX 86 Systems
- iRMX 862** FORTRAN 86/88 Compiler for execution on iRMX 86 Systems
- iRMX 863** PL/M 86/88 Compiler for execution on iRMX 86 Systems
- ISBC 957B** iAPX 86 System Monitor and Micro-computer Development System Communications Link

Supported Hardware Products

ISBC® MULTIBUS® PRODUCTS

- iSBC 86/12A, 86/14, and 86/30 Single Board Computers
- iSBC 208 Flexible Disk Controller

PERIPHERAL DEVICE

- CRT** — RS232 at 9600 Baud
- Printer** — Centronics-type Parallel Interface
- Diskettes** — 2 to 4 Single- or Double-Density, Single- or Double-Sided

Memory Requirements

- 200K Bytes to support applications less than 16K Bytes.
- 384K Bytes to support Intel's PASCAL 86 Compiler.
- 256K Bytes to support Microsoft's Basic Interpreter and a 32K Byte user program and data space.

Reference Material

- iRMX 86 Operating System Data Sheet (210330)

- Getting Started with the iRMX 86 System (144340-001) (Included in PC System Package)
- Introduction to the iRMX 86 Operating System (9803124-03)
- iRMX 86 Installation Guide (9803125-04)
- iRMX 86 Configuration Guide (9803126-04)
- iRMX 86 NUCLEUS Reference Manual (9803122-03)
- iRMX 86 Terminal Handler Reference Manual (143324-01)
- iRMX 86 Debugger Reference Manual (143323-01)
- iRMX 86 Basic I/O System Reference Manual (9803123-04)
- iRMX 86 Loader Reference Manual (143318-01)
- iRMX 86 Extended I/O System Reference Manual (143318-001)
- iRMX 86 Human Interface Reference Manual (9803202-002)
- iRMX 86 System Programmer's Reference Manual (142721-003)
- Guide to Writing Device Drivers for the iRMX 86 and iRMX 88 I/O Systems (142926-003)
- iRMX 86 Programming Techniques (142982-002)
- User's Guide for the iSBC 857B iAPX 86,88 Interface and Execution Package (143979-002)
- iRMX 86 Disk Verification Utility Reference Manual (144133-001)
- iRMX 86 Pocket Reference (142861-002)
- Edit Reference Manual (143587-001)
- Runtime Support Manual for iAPX 86,88 Applications (121776-001)
- Guide to Using iRMX 86 Languages (143907-001)
- Reference material may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

Training Courses

- Introduction to the iRMX 86 Operating System
- iRMX 86 I/O System Concepts

ORDERING INFORMATION

The iRMX 86 PC System is provided on a double-density, iRMX 86 compatible system diskette (format type E). The iRMX 86 PC System is shipped with a comprehensive users' manual ("Getting Started With The iRMX 86 System), Bootloader and Monitor EPROMs, and the complete iRMX 86 Interface Libraries contained on a second diskette. A full year of Intel Support Level D (Software Problem Report Service) is included. This Intel copyrighted system is licensed as a single-use software product as defined by Intel's Master Software Licenses.

Order Code Description

RMX 86PC E Complete Preconfigured iRMX 86 Operating System with interface libraries, bootstrap monitor, and user documentation.

iRMX 86 Extended IO System Reference Manual (143318-001)
iRMX 86 Human Interface Reference Manual (9803202-002)
iRMX 86 System Programmer's Reference Manual (143321-003)
Guide to Writing Device Drivers for the iRMX 86 and iRMX 86 I/O Systems (143326-003)
iRMX 86 Programming Techniques (143328-002)
User's Guide for the iSBC 857B iAPX 86/88 Interface and Execution Package (143376-002)
iRMX 86 Disk Verification Utility Reference Manual (144133-001)
iRMX 86 Pocket Reference (143381-002)
iRMX 86 Reference Manual (143387-001)
Runtime Support Manual for iAPX 86/88 Applications (121776-001)
Guide to Using iRMX 86 Languages (143307-001)
Reference material may be ordered from any Intel sales representative, distributor, or from Intel Literature Department, 3085 Bowers Avenue, Santa Clara, CA 95051.

Training Courses

Introduction to the iRMX 86 Operating System
iRMX 86 I/O System Concepts

SPECIFICATIONS

Optional Intel Software Products

iRMX 86 Fully configurable iRMX 86 Realtime Operating System
iRMX 86 Development Utilities Package including the iAPX 86 and 88 Linker, Loader, and Macro Assembler, Librarian, and the iRMX 86 Editor
iRMX 86 PASCAL 85/88 Compiler for execution on iRMX 86 Systems
iRMX 86 FORTRAN 85/88 Compiler for execution on iRMX 86 Systems
iRMX 86 PLM 85/88 Compiler for execution on iRMX 86 Systems
iSBC 857B iAPX 86 System Monitor and Microcomputer Development System Communications Link

Supported Hardware Products

iSBC MULTIBUS PRODUCTS

iSBC 8612A, 8614, and 8610 Single Board Computers
iSBC 208 Flexible Disk Controller

PERIPHERAL DEVICE

CRT — RS232 at 9600 baud
Printer — Centronics-type Parallel Interface
Diskettes — 5 to 4 Single- or Double-Density, Single- or Double-Sided

Memory Requirements

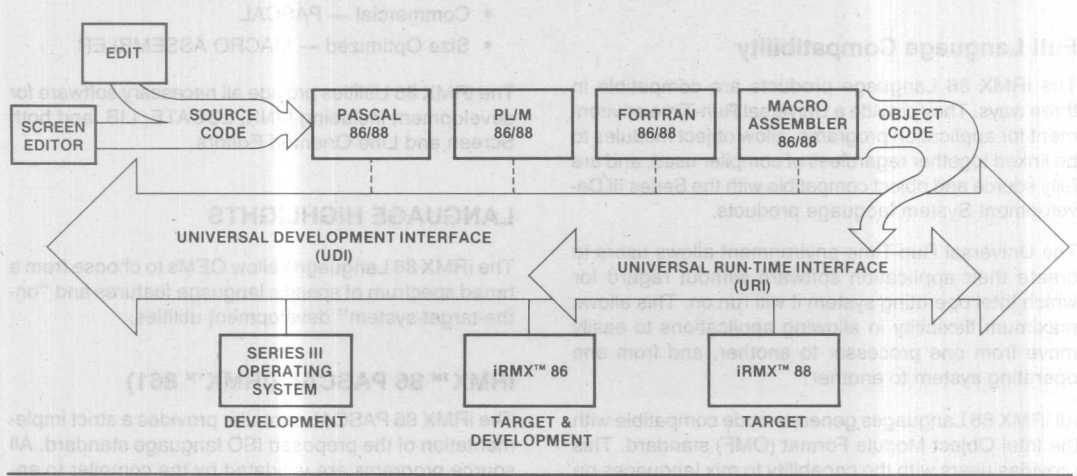
256K Bytes to support applications less than 16K Bytes
284K Bytes to support Intel's PASCAL 86 Compiler
256K Bytes to support Microsoft's Basic Interpreter and a 32K Byte user program and data space

Reference Material

iRMX 86 Operating System Data Sheet (210330)

IRMX™ 86 LANGUAGES AND UTILITIES

• **Systems Programs** — PL/M



© INTEL CORPORATION, 1982

FEATURES AND BENEFITS

Major features of the iRMX 86 Languages and Utilities and their benefits include:

Target System Development

OEMs may accomplish application software development on their target application hardware. This provides the greatest possible utility of the OEM's application system. Additionally, OEMs may provide an entirely new dimension in flexibility — reprogrammability of his system by the end-user.

The iRMX 86 Languages run directly on the iRMX 86 Operating System. This reduces the user's learning curve, since users only need to learn one operating system interface.

New Technology Languages

The iRMX 86 Languages provide OEMs with the newest programming languages available. iRMX 86 PASCAL is fully compatible with the proposed ISO language standard. iRMX 86 FORTRAN provides the majority of the ANS 77 FORTRAN language features, including IF-THEN-ELSE, Zero Case Do-Loops, and Direct Access I/O. The complex arithmetic capability, the only missing major feature, will be supplied in a later release.

Efficient Application Code

The iRMX Language products are optimized to provide a maximum efficiency in object code generation. This provides users with the smallest, fastest programs which a high level language can generate.

Full Language Compatibility

The iRMX 86 Language products are compatible in three ways. They provide a Universal Run-Time environment for application programs, allow object modules to be linked together regardless of compiler used, and are fully source and object compatible with the Series III Development System language products.

The Universal Run-Time environment allows users to create their application software without regard for which Intel operating system it will run on. This allows maximum flexibility in allowing applications to easily move from one processor to another, and from one operating system to another.

All iRMX 86 Languages generate code compatible with the Intel Object Module Format (OMF) standard. This provides users with the capability to mix languages on a single application system. In this way a user can select

exactly the right language tool for specific parts of the application rather than a project as a whole.

Intel Series III Development System languages provide all of the same benefits described, allowing users to develop their software on a system optimized for program development and then easily move it to the final system for test, debug and minor redevelopment.

Full REALMATH Support

The iRMX 86 Languages support the REALMATH floating point standard. This allows users of all iRMX 86 languages to access the iAPX 86/20 or iAPX 88/20 Numeric Data Processors or iSBC 337 MULTIMODULE board. These numeric processors offer over 100 times greater performance than comparable software-implemented algorithms, and reduce the system memory requirements by at least 16 KB. The REALMATH standard (proposed IEEE standard) provides universal consistency in results of numeric computations. The iRMX 86 Languages provide efficient object code generation and access to the highest performance floating point package available on microcomputers.

Complete Set of Languages and Utilities

The iRMX 86 Languages and Utilities products offer a broad selection of modern, highly efficient language products and a complete set of target system utility software.

iRMX 86 Languages allow you to select the correct language for your application.

- Technical — FORTRAN or PASCAL
- Systems Programming — PL/M
- Commercial — PASCAL
- Size Optimized — MACRO ASSEMBLER

The iRMX 86 Utilities provide all necessary software for development including LINK, LOCATE, LIB, and both Screen and Line Oriented Editors.

LANGUAGE HIGHLIGHTS

The iRMX 86 Languages allow OEMs to choose from a broad spectrum of specific language features and "on-the-target-system" development utilities.

iRMX™ 86 PASCAL (iRMX™ 861)

The iRMX 86 PASCAL compiler provides a strict implementation of the proposed ISO language standard. All source programs are validated by the compiler to ensure its conformance to the standard. Many extensions

to the language are available which allow PASCAL programs to be written specifically for microcomputers. Features such as separate module compilation and iAPX 86/20, 88/20 Numeric Data Processor support are just a few of the extensions available. The ISO standard "source evaluator" can be switched off to accept these extensions. For more information on iRMX 86 PASCAL features, see the PASCAL 86/88 Software Package data sheet (order number: 400670).

iRMX™ 86 FORTRAN (iRMX™ 862)

The iRMX 86 FORTRAN compiler provides users total compatibility with existing FORTRAN 66 language-generated code, plus many new language features provided by the FORTRAN 77 language standard. These new features offer FORTRAN programmers many new capabilities, including "IF-THEN-ELSE", random access I/O and character variables. For a more detailed explanation of iRMX 86 FORTRAN, see the FORTRAN 86/88 Software Package data sheet (order number: 400630).

iRMX™ 86 PL/M (iRMX™ 863)

The iRMX 86 PL/M compiler provides users with a powerful, microcomputer-oriented system programming language. The PL/M 80 Language was introduced in 1976 by Intel. It was the first microcomputer-oriented, block structured, high-level language available. Since 1976, thousands of users, shipping over millions of microcomputer-based systems, have generated their software with PL/M 80 and PL/M 86.

iRMX 86 PL/M 86 is a compatible superset of PL/M 80 which offers easy portability of software. For more information about iRMX 86 PL/M, see the PL/M 86/88 Software Package data sheet (order number: 402175).

iRMX™ 86 Screen Editor (iRMX™ 864)

The iRMX 86 Screen Editor provides users with a menu-driven, easy-to-learn Text-Editor. The user of the iRMX 864 Screen Editor is guided through all steps of the editing process by a menu which is always displayed on the CRT screen. By keeping the menu of commands always in view, even infrequent users of the Editor are able to edit text quickly and efficiently.

The iRMX 864 Screen Editor allows users to edit two files during a single session. This allows easy transferring of text between files and use of existing material in the creation of new files.

The process of creating "macros", strings of commands which are used frequently, is very simple. All you have to do is use the commands. The iRMX 864 Screen Editor remembers them and allows you to re-use them or store them for future use.

The iRMX 864 Screen Editor is also fully compatible with the iMDX 334 Alter Text Editor.

iRMX™ 86 Utilities (iRMX™ 860)

iRMX™ 86 EDIT

The iRMX 86 EDIT program provides the users with a powerful, sophisticated, line-oriented editing facility. EDIT delivers a range of capability suitable for novice users as well as advanced capabilities for sophisticated users. Its key features include a macro processor capable of creating and executing complex strings of commands, which ease the editing chore as well as defining blocks of texts which may be included anywhere in the text file. EDIT offers variable command sourcing, symbolic line numbering and reference by symbol.

The facilities of EDIT allow users to create, maintain, and manipulate extensive libraries of source code with minimal effort.

iRMX™ 86 LINK/LOCATE

The iRMX 86 LINK program connects object modules which have been individually compiled into a single, relocatable object module. The input object code may have been produced by any Object Module Format-compatible compiler. Output object modules may be recombined into larger object modules, allowing work from a large programming staff to be easily integrated into an application system.

The iRMX 86 LOCATE program maps the relocatable object code into the iAPX 86/88 memory segments. Modules may be targeted for specific memory types. For example, those portions of the application which must be (P)ROM resident can be mapped directly to the appropriate address range.

Both iRMX 86 LINK and LOCATE provide listings of resultant memory and symbol maps for easy reference and simplified debug.

iRMX™ 86 LIBRARIAN

The iRMX 86 LIBRARIAN "Library manager" allows creation and maintenance of object module libraries. These libraries offer easy collection of related object code to reduce the overhead of maintaining many separate modules. Users may create new libraries, add and delete object modules, as well as list the contents of the library and their public symbols. Individual modules within the library will automatically be included in a total application system by the iRMX 86 LINK program.

SPECIFICATIONS

Required Hardware

Any iAPX 86/88-based or iSBC 86/88-based system or System 86/88 capable of running the iRMX 86 Operating System, Release 4 or later.

- In addition to the memory required for iRMX 86, the language and utility products will need 140KB
- Two iRMX 86-compatible floppy disks or one hard disk
- One 8" single-density floppy disk drive for distribution of software
- System console device

Optional Hardware

iAPX 86/20, iAPX 88/20 or iSBC 337 Numeric Data Processors for support of the REALMATH standard.

Required Software

The iRMX 86 Operating System, Release 4 or later including the nucleus, basic I/O system, extended I/O system and human interface.

Documentation Packages

Package	Manual Included
RMX 860	— EDIT Manual (143587) — Guide to Using iRMX Languages (143907) — iAPX 86/88 Family Utilities (121616) — Macro Assembly Language (121627) — Macro Assembly Language Operations (121628)
RMX 861	— PASCAL 86 User's Guide (121539)
RMX 862	— FORTRAN 86 User's Guide (121570)
RMX 863	— PL/M 86 User's Guide (121636)
RMX 864	— TX User Screen Editors User's Guide (220125)

ORDERING INFORMATION

The products listed below require the signing of an Intel End User Software License Agreement or OEM Software License Agreement. All DRO products below include 1 year of update service. All products below are support category B.

Part Number Description

iRMX™ 86 Utility Package	(EDIT, LINK, LOCATE, LIB, MACRO ASSEMBLER)
RMX 860 DRO	8" single-density, single-sided iRMX 86 compatible diskettes
RMX 860 DWX	Update service on 8" single-density, single-sided diskettes

iRMX™ 86 PASCAL 86/88

RMX 861 DRO	8" single-density, single-sided iRMX 86 compatible diskettes
-------------	--

RMX 861 DWX	Update service on 8" single-density, single-sided diskettes
-------------	---

iRMX™ 86 FORTRAN 86/88

RMX 862 DRO	8" single-density, single-sided iRMX 86 compatible diskettes
RMX 862 DWX	Update service on 8" single-density, single-sided diskettes

iRMX™ 86 PL/M 86/88

RMX 863 DRO	8" single-density, single-sided iRMX 86 compatible diskettes
RMX 863 DWX	Update service on 8" single-density, single-sided diskettes

iRMX™ 86 Screen Editor

RMX 864 DRO	8" single-density, single-sided iRMX 86 compatible diskettes
RMX 864 DWX	Update service on 8" single-density, single-sided diskettes



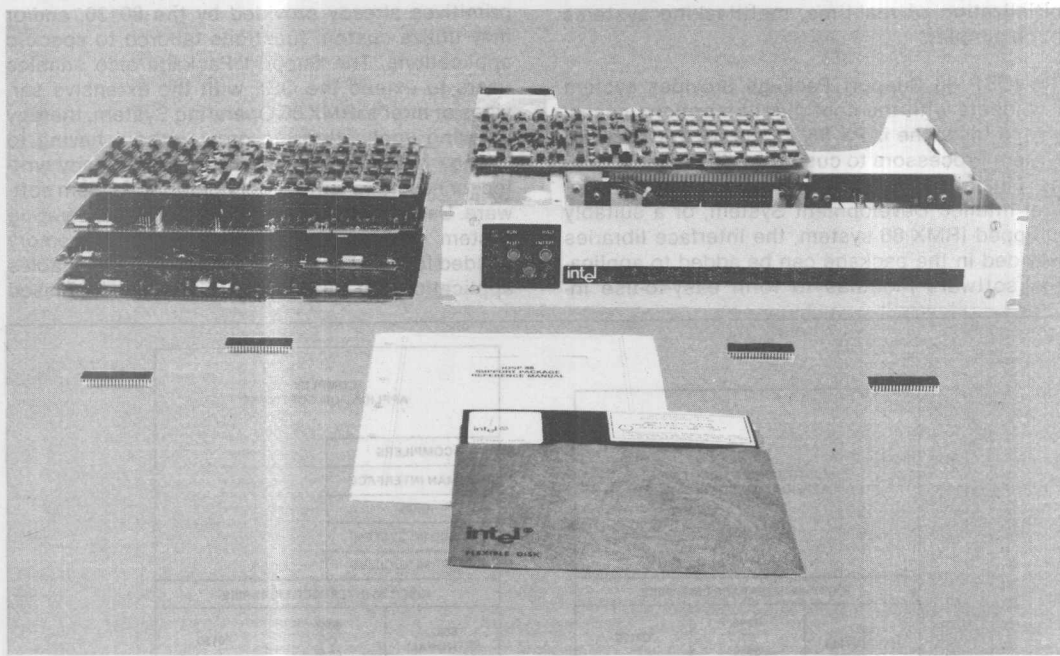
INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 • (408) 987-8080

iOSP™ 86

iAPX 86/30 AND iAPX 88/30 SUPPORT PACKAGE

- Development and run-time support for iAPX 86/30 and 88/30 Operating System Processors
- Total iRMX™ 86 Operating System software compatibility
- Extendable with iRMX™ 86 Operating System calls
- Compatible with Intel® PL/M 86/88, PASCAL 86/88, FORTRAN 86/88, and iAPX 86/88 ASSEMBLER
- Supports (P)ROM or RAM based system
- Complete system initialization aids
- Complete system configuration aids

The Intel iOSP 86 Support Package for the iAPX 86/30 and 88/30 Operating System Processors contains a comprehensive set of easy-to-use tools necessary to develop (P)ROM or RAM-based applications that use the 80130 Operating System Firmware component. All of the system initialization and run-time facilities are provided in libraries that may be configured to specific requirements, and linked to application programs written in either iAPX 86 or iAPX 88 Assembler or a high level programming language such as PASCAL 86 and PL/M 86. The iOSP 86 Package provides users with the basic initialization and interface routines needed to build application software based on the fundamental operating system functions of the iAPX 86/30 and 88/30 Operating System Processors. The iOSP 86 Package also enables users to add higher level I/O functions from the fully compatible iRMX 86 Operating System, or to form custom, real-time systems.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE, iOSP and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

FUNCTIONAL DESCRIPTION

The iAPX 86/30 and iAPX 88/30 Operating System Processors (OSPs) provide an easy-to-use foundation on which many real-time applications may be built. They provide the functions and system support needed to implement both simple and complex applications that require multiple tasks to run concurrently (see Figure 1). These services are made possible by the addition of the five new data types integrated into the 80130 Operating System Firmware (OSF) component. The 80130 OSF extends the basic data types of the CPU (integer, byte, character, etc.) by adding new system data types (JOB's, TASK's, MAILBOX's, SEGMENT's, and REGION's), and extensive timer, interrupt, memory, and error management designed to give real-time response to multitasking and multi-programming applications. As shown in the second half of the figure, other operating system functions such as mass storage I/O services and an easy-to-use Human Interface can be added easily, by using modules from the complete operating system services of the iRMX 86 Operating System. The iOSP 86 Support Package provides both an interface between application software and the Operating System Processors, and development tools designed to make the implementation and initialization of real-time, multitasking systems much simpler.

The iOSP 86 Support Package provides system developers with the configuration options necessary to tailor the iAPX 86/30 and 88/30 Operating System Processors to custom applications. Using the Linking and Locating facilities of either an Intel Inteltec Development System, or a suitably equipped iRMX 86 system, the interface libraries provided in the package can be added to application software modules to form easy-to-use in-

italization routines. They also form a simple interface between application software and the operating system primitives of the 80130 OSF component. The various configuration options include:

Memory and I/O Addressing

The 80130 OSF requires a 16K byte block of memory address space to be reserved for accessing internal functions. The iOSP 86 Support Package is used to specify the base address of the 80130 and the beginning of the initialization routines.

All Interrupt and Timer management of the OSF is controlled via a reserved 16-byte I/O address block that may be selected by the user. In addition, from 1 to 7 slave 8259A interrupt controllers can be specified in order to provide the system with up to 57 priority interrupt sources. The OSF baud rate generator may also be configured to support an optional terminal interface.

Extending the 80130 OSF

The 80130 OSF allows users to add their own operating system extensions. These extensions may take advantage of the detailed and efficient intertask communication and synchronization primitives already provided by the 80130, and/or may utilize custom functions tailored to specific applications. The Support Package also enables users to extend the OSF with the extensive services of Intel's iRMX 86 Operating System, thereby allowing applications to grow without having to change or alter application software already written, or having to write other operating system software. Use of the 80130 with the iRMX 86 Operating system greatly reduces the amount of memory needed for the iRMX 86 Nucleus layer, and enables applications to take advantage of the increased

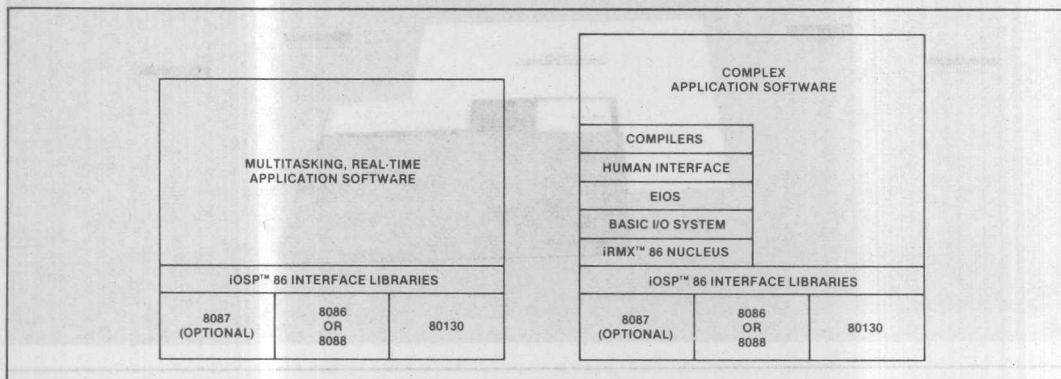


Figure 1. Structure of Typical Systems

ORDERING INFORMATION

Each of the ordering options listed below include all the necessary initialization and interface procedures needed to use the iAPX 86/30 and iAPX 88/30 Operating System Processors. Purchase of the iOSP 86 Package requires verification of an Intel Master Software License. Each package also includes an iOSP 86 User's Manual (Document Number 143331), and a one-year update service.

[illegible]

OSP 86 A	iOSP 86 Support Package contained on an ISIS-II compatible, single density diskette.
OSP 86 B	iOSP 86 Support Package contained on an ISIS-II compatible, double density diskette.
OSP 86 E	iOSP 86 Support Package contained on an iRMX 86 format, double density diskette.

I. INTRODUCTION

The iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package contains the hardware and software required to interface an iSBC 86/12 Single Board Computer with an Intellec Microcomputer Development System. The iSBC 957 package gives the 8086 user the capability to develop software on an Intellec System and then debug this software on an iSBC 86/12 board using a program download capability and an interactive system monitor. The 8086 user has all the capabilities of the Intellec system at his disposal and has the powerful iSBC 86/12 system monitor commands to use for debugging 8086 programs.

The iSBC 86/12 board is an Intel 8086 based processor board which, in addition to the processor, contains 32K bytes of dual port RAM, sockets for up to 16K bytes of ROM/EPROM, a serial I/O port, 24 parallel I/O lines, 2 programmable counters, 9 levels of vectored priority interrupts, and an interface to the MULTIBUS™ system bus. The iSBC 957 package consists of monitor EPROMs for the iSBC 86/12 board, Loader software for the Intellec system, four (4) cable assemblies, assorted line drivers and terminators, and signal adapters. The iSBC 957 package provides the capability of downloading and uploading program and data blocks between an iSBC 86/12 board and an Intellec system. In addition, monitor commands and displays may be input and viewed from the Intellec system console. The iSBC 957 package, when used with the iSBC 86/12 board and an Intellec Microcomputer Development System, provides the user with the capability to edit, compile or assemble, link, locate, download, and interactively debug programs for the 8086 processor. The iSBC 957 package and the iSBC 86/12 board form an excellent "execution vehicle" for users developing software for the 8086 processor regardless of whether the users are 8086 component users or iSBC 86/12 board users. Using the iSBC 957 package 8086 programs may be debugged at the full 5 MHz speed of the processor. The recommended hardware for the execution vehicle is an iSBC 660 system chassis with an 8 card slot backplane and power supply, an iSBC 032 32K byte RAM memory board, the iSBC 957 package, and the iSBC 86/12 board.

This application note will describe how the iSBC 957 package may be used to develop and debug 8086 programs. First a description of the iSBC 86/12 board will be presented. Readers familiar

with the iSBC 86/12 board may want to skip this section. Next follows a detailed description of the iSBC 957 package and the iSBC 86/12 system monitor commands. A program example of a matrix multiplication routine will then be presented. This example will contain both assembly language and PL/M-86 procedures. The steps required to compile, assemble, link, locate and debug the program code will be explained in detail. A typical debugging session using the iSBC 86/12 system monitor will be presented.

II. THE iSBC™ 86/12 SINGLE BOARD COMPUTER

The iSBC 86/12 Single Board Computer, which is a member of Intel's complete line of iSBC 80/86 computer products, is a complete computer system on a single printed-circuit assembly. The iSBC 86/12 board includes a 16-bit central processing unit (CPU), 32K bytes of dynamic RAM, a serial communications interface, three programmable parallel I/O ports, programmable timers, priority interrupt control, MULTIBUS control logic, and bus expansion drivers for interface with other MULTIBUS-compatible expansion boards. Also included is dual port control logic to allow the iSBC 86/12 board to act as a slave RAM device to other MULTIBUS masters in the system. Provision is made for user installation of up to 16K bytes of read only memory. Figure 1 contains a block diagram of the iSBC 86/12 board and in Appendix A is a simplified logic diagram of the iSBC 86/12 board.

Central Processing Unit

The central processor for the iSBC 86/12 board is Intel's 8086, a powerful 16-bit H-MOS device. The 225 sq. mil chip contains 29,000 transistors and has a clock rate of 5MHz. The architecture includes four (4) 16-bit byte addressable data registers, two (2) 16-bit memory base pointer registers and two (2) 16-bit index registers, all accessed by a total of 24 operand addressing modes for complex data handling and very flexible memory addressing.

Instruction Set—The 8086 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions. The instruction set of the 8086 is a functional superset of the 8080A/8085A family and with

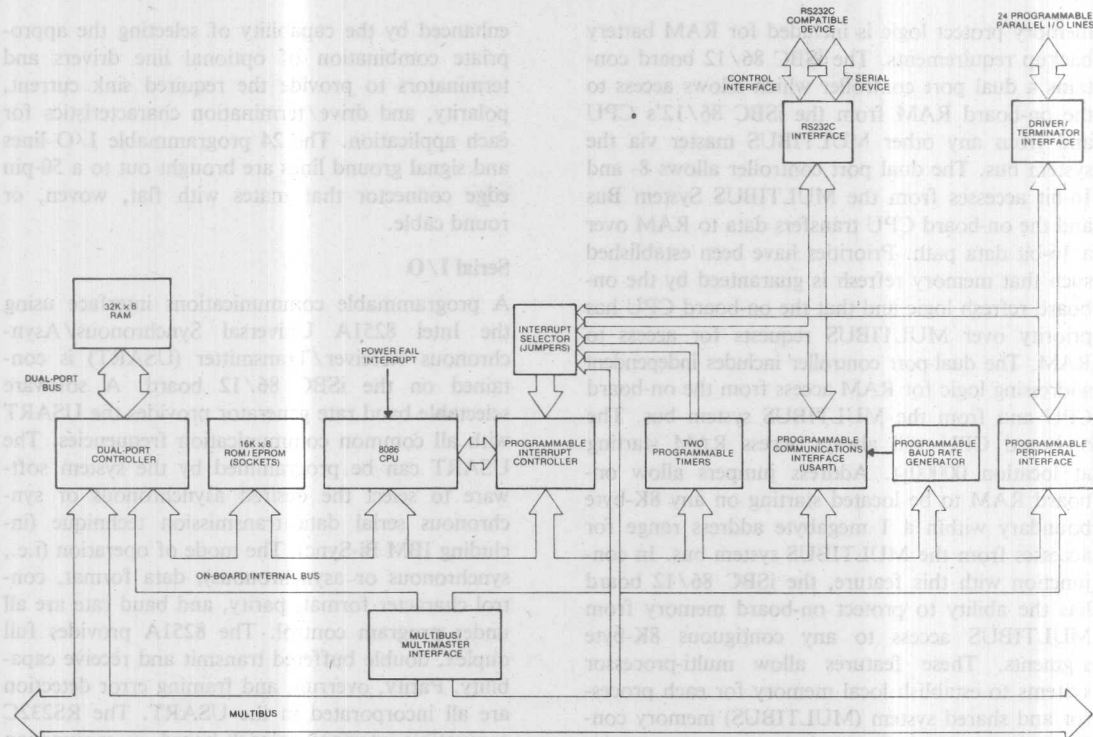


Figure 1. iSBC™ 86/12 Single Board Computer Block Diagram

available software tools, programs written for the 8080A/8085A can be easily converted and run on the 8086 processor.

Architectural Features — A 6-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 1.2 μ sec minimum instruction cycle to 400 nsec by having the instruction already in the queue.

The stack oriented architecture facilitates nested sub-routines and co-routines, reentrant code and powerful interrupt handling. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K-bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions.

Bus Structure

The iSBC 86/12 board has an internal bus for communicating with on-board memory and I/O options, a system bus (the MULTIBUS) for referencing additional memory and I/O options, and the dual-port bus which allows access to RAM from the on-board CPU and the MULTIBUS System Bus. Local (on-board) accesses do not require MULTIBUS communication, making the system bus available for use by other MULTIBUS masters (i.e. DMA devices and other single board computers transferring to additional system memory). This feature allows true parallel processing in a multiprocessor environment. In addition, the MULTIBUS interface can be used for system expansion through the use of other 8- and 16-bit iSBC computers, memory and I/O expansion boards.

RAM Capabilities

The iSBC 86/12 board contains 32K-bytes of dynamic read/write memory. Power for the on-board RAM and refresh circuitry may be optionally provided on an auxiliary power bus, and

memory protect logic is included for RAM battery backup requirements. The iSBC 86/12 board contains a dual port controller which allows access to the on-board RAM from the iSBC 86/12's CPU and from any other MULTIBUS master via the system bus. The dual port controller allows 8- and 16-bit accesses from the MULTIBUS System Bus and the on-board CPU transfers data to RAM over a 16-bit data path. Priorities have been established such that memory refresh is guaranteed by the on-board refresh logic and that the on-board CPU has priority over MULTIBUS requests for access to RAM. The dual-port controller includes independent addressing logic for RAM access from the on-board CPU and from the MULTIBUS system bus. The on-board CPU will always access RAM starting at location 00000H. Address jumpers allow on-board RAM to be located starting on any 8K-byte boundary within a 1 megabyte address range for accesses from the MULTIBUS system bus. In conjunction with this feature, the iSBC 86/12 board has the ability to protect on-board memory from MULTIBUS access to any contiguous 8K-byte segments. These features allow multi-processor systems to establish local memory for each processor and shared system (MULTIBUS) memory configurations where the total system memory size (including local on-board memory) can exceed 1 megabyte without addressing conflicts.

EPROM/ROM Capabilities

Four sockets are provided for up to 16K-bytes of non-volatile read only memory on the iSBC 86/12 board. Configuration jumpers allow read only memory to be installed in 2K, 4K, or 8K increments.

On-board ROM is accessed via 16 bit data paths. System memory size is easily expanded by the addition of MULTIBUS compatible memory boards available in the iSBC 80/86 family.

Parallel I/O Interface

The iSBC 86/12 board contains 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral Interface. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports.

Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further

enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector that mates with flat, woven, or round cable.

Serial I/O

A programmable communications interface using the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 86/12 board. A software selectable baud rate generator provides the USART with all common communication frequencies. The USART can be programmed by the system software to select the desired asynchronous or synchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The RS232C compatible interface on each board, in conjunction with the USART, provides a direct interface to RS232C compatible terminals, cassettes, and asynchronous and synchronous modems. The RS232C command lines, serial data lines, and signal ground line are brought out to a 26 pin edge connector that mates with RS232C compatible flat or round cable. The iSBC 530 teletypewriter adapter provides an optically isolated interface for those systems requiring a 20 mA current loop. The iSBC 530 adapter may be used to interface the iSBC 86/12 board to teletypewriters or other 20 mA current loop equipment.

Programmable Timers

The iSBC 86/12 board provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller and to the I/O line drivers associated with

the 8255A Programmable Peripheral Interface, or may be routed as inputs to the 8255A chip. The gate/trigger inputs may be routed to I/O terminators associated with the 8255A or as output connections from the 8255A. The third interval timer in the 8253 provides the programmable baud rate generator for the iSBC 86/12 RS232C USART serial port. In utilizing the iSBC 86/12, the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given time delay or count is needed, software commands to the programmable timers/counters select the desired function.

The contents of each counter may be read at any time during system operation with simple read operations for event counting applications, and special commands are included so that the contents can be ready "on the fly".

MULTIBUS™ and Multimaster Capabilities

The MULTIBUS system bus features asynchronous data transfers for the accommodation of devices with various transfer rates while maintaining maximum throughput. Twenty address lines and sixteen separate data lines eliminate the need for address/data multiplexing/demultiplexing logic used in other systems, and allow for data transfer rates up to 5 megawords/sec. A failsafe timer is included in the iSBC 86/12 board which can be used to generate an interrupt if an addressed device does not respond within 6 msec.

Multimaster Capabilities — The iSBC 86/12 board is a full computer on a single board with resources capable of supporting a great variety of OEM system requirements. For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication over the system bus), the iSBC 86/12 board provides full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 86/12 boards or other bus masters, including iSBC 80 family MULTIBUS compatible 8-bit single board computers, to share the system bus in serial (daisy chain) priority fashion, and up to 16 masters to share the MULTIBUS with the addition of an external priority network. The MULTIBUS arbitration logic operates synchronously with a MULTIBUS clock (provided by the iSBC 86/12 board or optionally provided directly from the MULTIBUS System Bus) while data is transferred via a handshake between the master and slave modules. This

allows different speed controllers to share resources on the same bus, and transfers via the bus proceed asynchronously. Thus, transfer speed is dependent on transmitting and receiving devices only. This design prevents slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed direct memory access (DMA) operations, and high speed peripheral control, but are by no means limited to these three.

Interrupt Capability

The iSBC 86/12 board provides 9 vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line which is directly tied to the 8086 CPU. This interrupt cannot be inhibited by software and is typically used for signalling catastrophic events (e.g., power failure).

The Intel 8259A Programmable Interrupt Controller (PIC) provides vectoring for the next eight interrupt levels.

The PIC accepts interrupt requests from the programmable parallel and serial I/O interfaces, the programmable timers, the system bus, or directly from peripheral equipment. The PIC then determines which of the incoming requests is of the highest priority, determines whether this request is of higher priority than the level currently being serviced, and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. The PIC generates a unique memory address for each interrupt level. These addresses contain unique instruction pointers and code segment offset values (for expanded memory operation) for each interrupt level. In systems requiring additional interrupt levels, slave 8259A PIC's may be interfaced via the MULTIBUS system bus, to generate additional vector addresses, yielding a total of 65 unique interrupt levels.

Interrupt Request Generation — Interrupt requests may originate from 16 sources. Two jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface.

Two jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU or a character is ready to be transmitted.

A jumper selectable request can also be generated by each of the programmable timers. Eight additional interrupt request lines are available to the user for direct interface to user designated peripheral devices via the system bus, and two interrupt request lines may be jumper routed directly from peripherals via the parallel I/O driver/terminator section.

Power-Fail Control

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the iSBC 635 Power Supply or equivalent.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. High speed integer and floating point arithmetic capabilities may be added by using the iSBC 310 high speed mathematics unit. Memory may be expanded to 1 megabyte by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

III. THE iSBC™ 957 PACKAGE

The iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package extends the software development capabilities of the Intellec Microcomputer Development systems to the Intel 8086 CPU. Programs for the 8086 may be written in PL/M-86 and/or assembly language and compiled or assembled on the Intellec system. These programs may then be downloaded from an Intellec ISIS-II disk file to the iSBC 86/12 board for execution and debug. The programs will execute at the full 5 MHz clock rate of the 8086 CPU with no speed degradation caused by the iSBC 957 hardware or software. Special communication software allows transparent access to the powerful interactive debug commands in the iSBC 86/12 monitor from the Intellec console terminal. These debug commands include single-step instruction execution, execution with breakpoints, memory and register displays, memory searches, comparison of two memory blocks and several other commands. After a debugging session, the debugged program code may be uploaded from the iSBC 86/12 board to an Intellec ISIS-II disk file.

The iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package consists of the following:

- a. Four Intel 2716 EPROMs which contain the system monitor program for the iSBC 86/12 board.
- b. An ISIS-II diskette containing loader software for execution in the Intellec which provides for communications between the user or an Intellec ISIS-II file and the iSBC 86/12 board. Also included on the diskette are a library of routines for system console I/O.
- c. Four cable assemblies used for transmitting commands, code and data between the iSBC 86/12 board and the Intellec system.
- d. An iSBC 530 adapter assembly which converts serial communications signals from current loop to RS232C.
- e. Line drivers and terminators used for the iSBC 86/12 parallel ports.
- f. A small printed circuit board which is plugged into an iSBC 86/12 receiver/terminator socket and is used when program code is downloaded or uploaded using the parallel cable.

iSBC™—Intellec™ Configurations

There are two distinct functional configurations for the iSBC 957 package; one configuration for the Intellec Series II, Models 220 or 230 development systems and another for the Intellec 800 series development systems.

Intellec Series II System Configurations

When used with Intellec Series II Model 220 or 230 systems, a set of cables are used to connect the serial I/O port edge connector on the iSBC 86/12 board and the SERIAL 1 output port on the Intellec system. This configuration is shown in Figure 2. How this system functions is explained in the following paragraphs.

The SERIAL 1 port on the Intellec Series II Model 220 or 230 system is an RS232 port which is designed for use with a data terminal. This port may be used on the Intellec system for interfacing to RS232 devices such as CRT terminals or printers. The serial ports on the iSBC 86/12 board and the Intellec systems are connected as shown in the Figure 2. The flat ribbon cable connected to the iSBC 86/12 board has an edge connector for connecting to the board on one end and a standard RS232 connector on the other end. The second cable, the RS232 Up/Down Load cable, has an RS232 connector on each end. This cable, however,

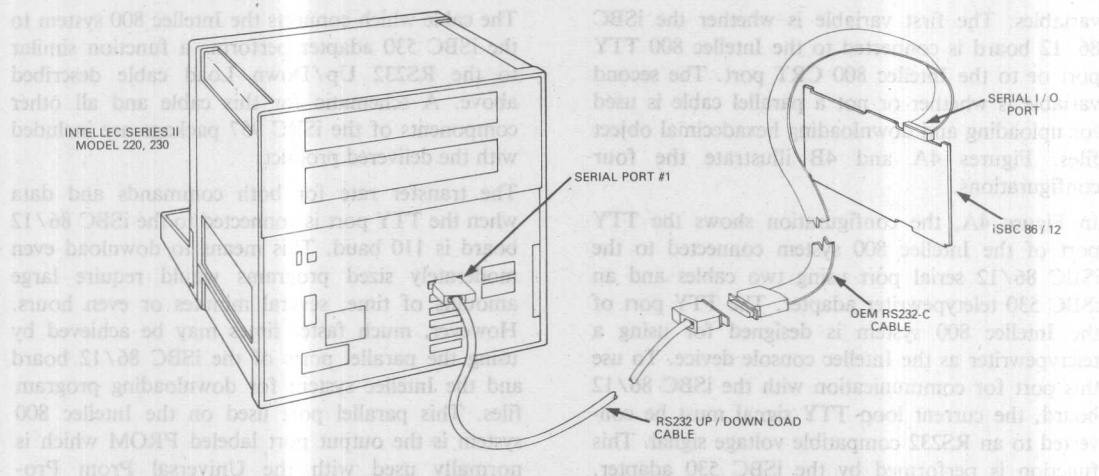


Figure 2. Inteltec™ Series II Model 220, 230—iSBC™ 86/12 Configuration

is not a standard cable with the RS232 signals bussed between identically numbered pins on each of the connectors. The schematic for the cable is shown in Figure 3. Note that the TXD (transmit data) and the RXD (receive data) and the RTS (ready to send) and the CTS (clear to send) signals have been crossed. This is done because both the Inteltec system and the iSBC 86/12 board are configured to act as data sets which are communicating with data terminals. Swapping these signals permits the units to communicate directly with no modifications to the Inteltec or iSBC 86/12 systems themselves.

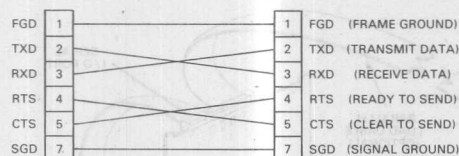


Figure 3. Inteltec™—iSBC™ 86/12 RS232 UP / DOWN LOAD Cable

The software in the Inteltec system accepts characters output from the iSBC 86/12 board through the Inteltec SERIAL 1 port. The software then outputs these characters on the CRT monitor built into the Inteltec Series II Model 220 or 230. In a similar fashion, characters input from the Inteltec key-

board are passed down the serial link to the iSBC 86/12 monitor program. The integrated CRT monitor and keyboard on the Inteltec system then becomes the "virtual terminal" of the iSBC 86/12 monitor program. If this were the only function of the iSBC 957 package, there would be no real benefit to the user. However, when the "virtual terminal" capability is combined with the capability to download and upload program code and data files between the Inteltec ISIS-II file system and the iSBC 86/12 board, a very powerful software development tool is realized. The software in the Inteltec system must examine the commands which are input from the keyboard and in the case of the LOAD and TRANSFER commands (see later sections for details on monitor commands), the software must open and read or write ISIS-II disk files.

Transfer rates using Inteltec Series II Model 220 or 230 system are 9600 baud when transferring hexadecimal object files to or from a disk file and 600 baud when transferring commands between the iSBC 86/12 board and the CRT monitor and keyboard. With a 9600 baud transfer rate, it is possible to load 64K bytes of memory in about four minutes.

Inteltec 800 System Configurations

The iSBC 957 package may be used with the Inteltec 800 system in four different configurations. These four configurations are determined by two

variables. The first variable is whether the iSBC 86/12 board is connected to the Intellec 800 TTY port or to the Intellec 800 CRT port. The second variable is whether or not a parallel cable is used for uploading and downloading hexadecimal object files. Figures 4A and 4B illustrate the four configurations.

In Figure 4A, the configuration shows the TTY port of the Intellec 800 system connected to the iSBC 86/12 serial port using two cables and an iSBC 530 teletypewriter adapter. The TTY port of the Intellec 800 system is designed for using a teletypewriter as the Intellec console device. To use this port for communication with the iSBC 86/12 board, the current loop TTY signal must be converted to an RS232 compatible voltage signal. This function is performed by the iSBC 530 adapter.

The cable which connects the Intellec 800 system to the iSBC 530 adapter performs a function similar to the RS232 Up/Down Load cable described above. A schematic for this cable and all other components of the iSBC 957 package are included with the delivered product.

The transfer rate for both commands and data when the TTY port is connected to the iSBC 86/12 board is 110 baud. This means to download even moderately sized programs would require large amounts of time, several minutes or even hours. However, much faster times may be achieved by using the parallel ports of the iSBC 86/12 board and the Intellec system for downloading program files. This parallel port used on the Intellec 800 system is the output port labeled PROM which is normally used with the Universal Prom Pro-

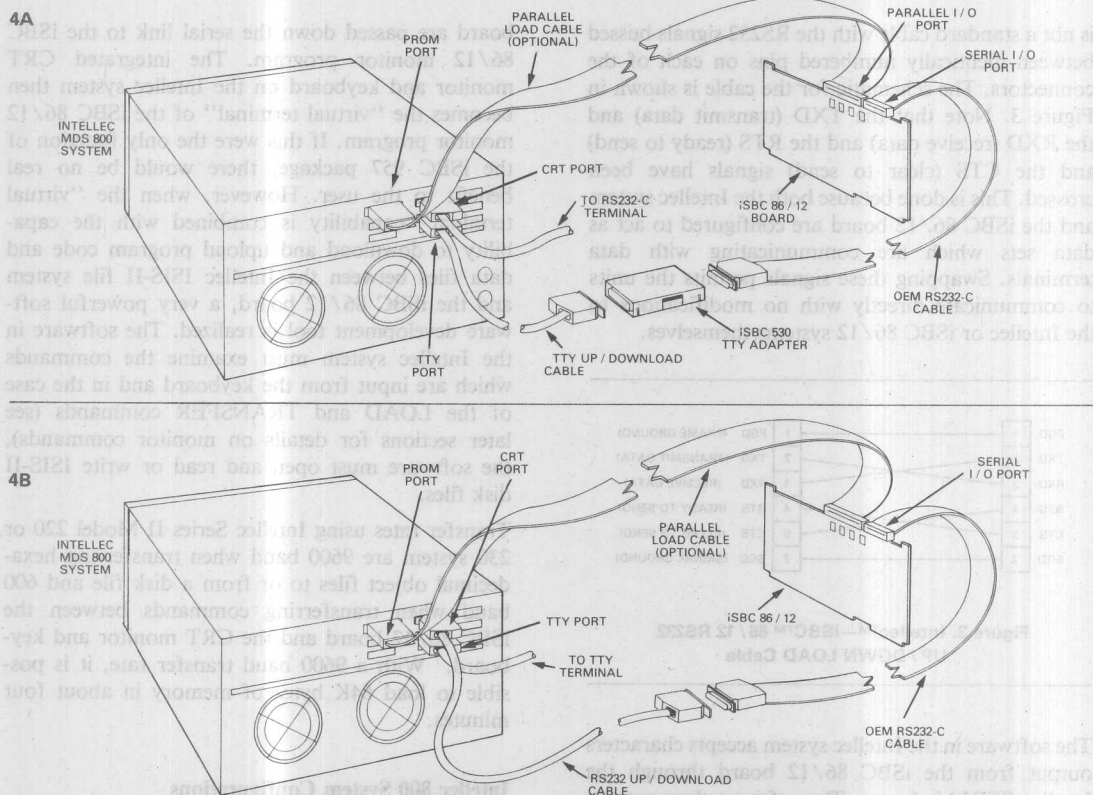


Figure 4A, 4B. Intellec™ 800—iSBC™ 86/12 Configurations

grammer. A cable is connected between the Intellec PROM port and the parallel I/O port, J1 of the iSBC 86/12 board. Parallel port B of the iSBC 86/12 board is used for the 8-bit byte transfers from the Intellec system to the iSBC 86/12 board, port A is used for the byte transfers from the iSBC 86/12 board to the Intellec system and port C is used for controlling the byte transfers. A special status adapter piggyback board must be inserted into a receiver/terminator socket of the iSBC 86/12 board. This status adapter circuit is required to provide the necessary handshaking signals from the iSBC 86/12 parallel ports to the Intellec PROM port.

The transfer rate achieved when downloading and uploading hexadecimal object files with the parallel cable is approximately 1,000 bytes per second. The time required to load 64K bytes of memory is approximately 2½ minutes.

Figure 4B shows a configuration with the Intellec 800 CRT port connected to the serial port of the iSBC 86/12 board. The TTY port of the Intellec 800 system is connected to a teletypewriter or some other current loop device to act as a system console. The optional parallel load cable is also shown. The cables used for this configuration are the same as those used with the Intellec Series II Configurations. Command transfer rates require 110 baud because the TTY port of the Intellec 800 system is used for communicating with the console device. However, hexadecimal object files can be loaded at 9600 baud since this operation uses only the Intellec to iSBC 86/12 RS232 link.

It is also possible to download files with the parallel cable, this mode being somewhat faster than the serial download mode (2½ minutes versus four minutes for 64K bytes of memory). Table I contains a summary of the command and memory transfer rates for each of the Intellec-iSBC 86/12 configurations.

Comparing the Intellec 800 configurations shown in Table 1 and in Figures 4A and 4B it should be noted:

1. Using the TTY port (Figure 4A) of the Intellec 800 system for communications with the iSBC 86/12 board (essentially) requires installation of the parallel cable and jumper modifications for downloading and uploading files, and thus, prevents the use of the parallel ports for other I/O functions.
2. Using the CRT port (Figure 4B) of the Intellec

800 system for communication with the iSBC 86/12 board provides for a fast serial download capability, thus freeing the parallel ports for other uses. However, this configuration requires a teletypewriter or a CRT capable of accepting a current loop input signal as the Intellec system console.

Table 1
COMMAND AND MEMORY TRANSFER RATES FOR
INTELLEC—iSBC™ 86/12 CONFIGURATIONS

	INTELLEC SERIES II 220/230 SERIAL PORT TO iSBC 86/12	INTELLEC 800 TTY PORT TO iSBC 86/12	INTELLEC 800 CRT PORT TO iSBC 86/12
Effective Command Rate	600 Baud	110 Baud	110 Baud*
Load / Transfer Rate			
Serial	9600 Baud	110 Baud	9600 Baud
Parallel	N/A	1K bytes/sec**	1K bytes/sec**
Approximate Time to load 64K bytes of memory			
Serial	4 minutes	5 hours	4 minutes
Parallel	N/A	2.5 minutes	2.5 minutes

*The actual baud rate of the Intellec—iSBC 86/12 link is 9600 baud, but the effective command rate is determined by the slower Intellec—console serial link.

**Transmission rate over the parallel link is determined by the speed of the two processors and is approximately 1K bytes per second.

IV. THE iSBC 957—iSBC 86/12 MONITOR PROGRAM

The iSBC 86/12 monitor program is an EPROM resident program which facilitates debugging of user written programs. The monitor program used in the iSBC 86/12 board with the iSBC 957 package is the same monitor program written to interface the iSBC 86/12 directly to an RS232C data terminal. When interfaced directly to a terminal, the iSBC 86/12 board functions in a stand-alone environment communicating directly with the user via the data terminal. A user may use the monitor for entering small programs in hexadecimal format, executing a program, examining registers and memory contents, etc.

To use the monitor program with an Intellec system, the proper cables must be installed and the iSBC 957 Loader program must be loaded into Intellec memory and executed. The Loader program is resident on a file named SBC861, and when executed, the Loader outputs a sign-on message. Next, the iSBC 86/12 monitor program must be started and the baud rate of the iSBC 86/12 to Intellec serial communications link must be determined. This is done by pressing the RESET switch on the chassis

Table 2
MONITOR COMMAND LIST

COMMAND	FUNCTION AND SYNTAX
L Load Hex Object File	Loads hexadecimal object file from Intellec into iSBC 86/12 memory using serial (S) or parallel (P) mode. L{S P}<filename>[,<bias addr>]<cr>
T Transfer Hex Object File	Transfers blocks of iSBC 86/12 memory to Intellec as a hex object file using serial (S) or parallel (P) mode. T{X S P}<start addr>,<end addr>,<filename>[,<exec addr>]<cr>
E Exit	Exits the loader program and returns to ISIS. E<cr>
N Single Step	Executes one user program instruction. N[<addr>],[<addr>],[*]<cr>
G Go	Transfers control of the 8086 CPU to the user program with up to 2 optional breakpoints. G[<start addr>],[<break 1 addr>][,<break 2 addr>]<cr>
S Substitute Memory	Displays/modifies memory locations in byte or word format. S{W}<addr>,[<new contents>],[*]<cr>
X Examine/Modify Register	Displays/modifies 8086 CPU registers. X[<reg>][[<new contents>],[*]<cr>
D Display Memory	Displays contents of a memory block in byte or word format. D{W}<start addr>,<end addr>]<cr>
M Move	Moves contents of a memory block. M<start addr>,<end addr>,<destination addr>]<cr>
C Compare	Compares two memory blocks. C<start addr>,<end addr>,<destination addr>]<cr>
F Find	Searches a memory block for a byte or word constant. F{W}<start addr>,<end addr>,<data>]<cr>
H Hex Arithmetic	Performs hexadecimal addition and subtraction. H<data 1>,<data 2>]<cr>
I Port Input	Inputs and displays byte or word data from input port. I{W}<port addr>,[*]<cr>
O Port Output	Outputs byte or word data to output port. O{W}<port addr>,<data>,[<data>]*<cr>

Syntax conventions used in the command structure are as follows:

- [A] indicates that "A" is optional
- [A]* indicates one or more optional iterations of "A"
- indicates that "B" is a variable
- {A|B} indicates "A" or "B"
- <cr> indicates a carriage return is entered

Numeric arguments can be expressed as a number, the contents of a register, or the sum or difference of numbers and register contents. Thus, addresses and data can be expressed as follows:

```

addr ::= [expr]:<expr>
expr ::= <number>|<register>|<expr> {+|-} <number>
       <expr> {+|-} <register>
register ::= AX|BX|CX|DX|SP|BP|SI|DI|CS|DS|SS|ES|IP|FL
number ::= <digit>|<digit>*<number>
digit ::= 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F

```

Numeric fields within arguments are entered as hexadecimal numbers. The valid range of numerical values is from 0000-FFFF. Larger numbers may be entered, but only the last four digits (or two in the case of byte values) are significant. Leading zeros may be omitted.

An address argument consists of a segment value and an offset value separated by a colon (:). If a segment value is not specified, the default segment value is the CS register value.

containing the iSBC 86/12 board and typing two "U"s on the Intellec console. The ASCII uppercase character U has a binary pattern of alternating ones and zeros, the iSBC 86/12 monitor uses this pattern to determine the baud rate of the serial link. After the baud rate has been determined, the monitor program outputs a sign-on message to the console. An example of loader program execution and monitor program initialization is shown below (user entered characters are underlined).

;F1:SBC861

ISIS-II iSBC 86/12 LOADER, Vx.x

(user resets iSBC 86/12 board and types two "U"s)
iSBC 86/12 MONITOR, Vy.y

The monitor prompts with a period "." when it is ready for a command. The user can then enter a command file, which consists of a one- or two-character command followed by zero, one, or more arguments. The command may be separated from the first argument by an optional single space; a single comma is required as a delimiter between arguments. The command line is terminated by a carriage return or a comma depending on the command, and no action takes place until the command terminator is sensed. The user can cancel a command before entering the command terminator by pressing any illegal key (e.g., rubout or Control-X).

Table 2 contains a summary of the loader and monitor commands. These commands will not be explained in detail; instead, the next section of the application note will show examples of using these loader and monitor commands. The iSBC 957 User's Guide referenced at the front of this document does, however, contain a complete description of each of the monitor and loader commands.

Table 3 contains a list of the 8086 hardware registers and abbreviations used by the monitor program.

Table 3
8086 CPU REGISTERS

REGISTER NAME	ABBREVIATION
Accumulator	AX
Base	BX
Count	CX
Data	DX
Stack Pointer	SP
Base Pointer	BP
Source Index	SI
Destination Index	DI
Code Segment	CS
Data Segment	DS
Stack Segment	SS
Extra Segment	ES
Instruction Pointer	IP
Flag	FL

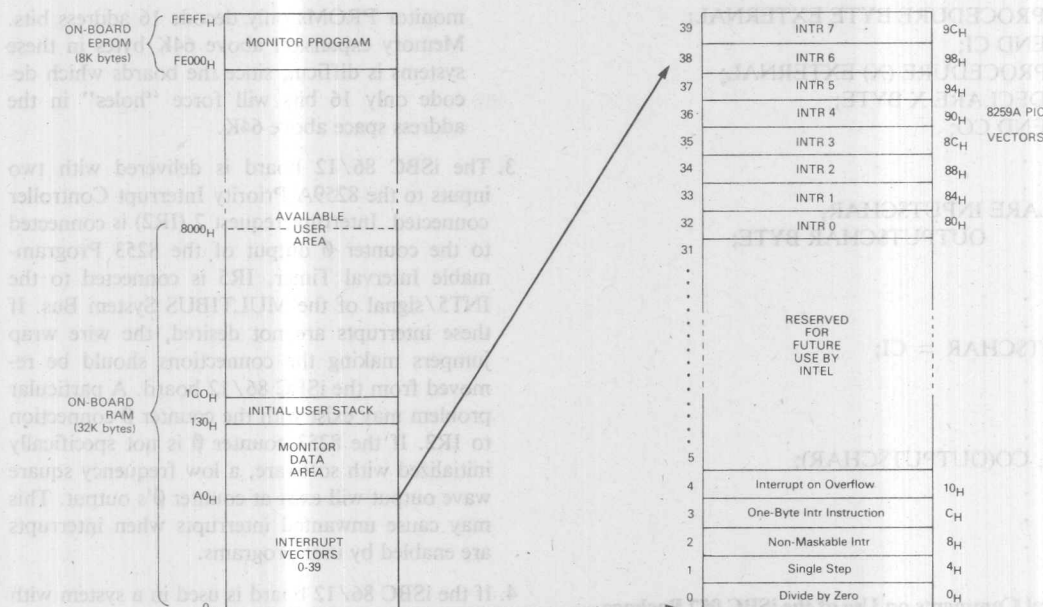


Figure 5. Memory Map of iSBC™ 86/12 Memory With Monitor Program

Figure 5 contains a memory map of the iSBC 86/12 memory with the monitor program. Note that the monitor uses the top 8K bytes of memory for its program code and the first 384 bytes of memory (locations 0 hex to 17F hex) for monitor and user stack, data and interrupt vectors. When the monitor program is reset, the segment registers, the IP and the flags are set to 0; and the SP is set to 01C0H allowing 64 bytes for the user's stack. If 64 bytes is not sufficient for the user's application program, the SP should be set to some other value. The monitor program sets the single-step, one-byte instruction trap and non-maskable interrupt vectors to monitor entry points. The monitor also sets the 8259A Priority Interrupt Controller to fully nested mode with level 0 at the highest priority and all interrupts unmasked. The eight interrupt vector addresses for the 8259A are also set to addresses in the monitor. User programs may change the 8259A interrupt vectors to interrupt service routine addresses within the user programs; it is not necessary for users to program the 8259A chip directly. When an interrupt occurs, control passes to either the monitor or directly to user code depending on the address stored in the vector location. When the monitor responds to an interrupt, it acknowledges the interrupt and displays the interrupt level, CS and IP register values and next instruction byte on

the system console (e.g., I = 3 @ 100:230F F5).

When a user requests a breakpoint with a "G" command, the monitor inserts the single byte instruction trap instructions (INT 3) in the location where the breakpoint is requested. It is also possible for the user to code an INT 3 instruction in his program. When a user coded INT 3 instruction is executed, the monitor will be re-entered and a line with the format @<CS Value>:<IP Value> <Instruction byte> will be displayed (e.g., @1200:3F02 F5).

Included on the diskette with the Loader program are two libraries containing I/O routines for the console. The library files are named SBCIOS.LIB and SBCIOL.LIB; they contain similar routines. The routines in SBCIOS.LIB are written to be called with intrasegment subroutine calls, a PL/M-86 module compiled with the "small" control generates this type of call. The routines in SBCIOL.LIB are written to be called with intersegment subroutine calls, a PL/M-86 module compiled with either the "medium" or "large" control generates this type of call.

The console input output routines, CI and CO, contained in the library should be used when performing character input and output on the console. Example PL/M-86 calls to the two routines are:

```

CI: PROCEDURE BYTE EXTERNAL;
END CI;
CO: PROCEDURE (X) EXTERNAL;
  DECLARE X BYTE;
  END CO;

```

```

DECLARE INPUT$CHAR,
        OUTPUT$CHAR BYTE;

```

```

INPUT$CHAR = CI;

```

```

CALL CO(OUTPUT$CHAR);

```

General Comments on Use of the iSBC 957 Package

1. If the iSBC 86/12 board is reset any time after the initial baud rate search, it is not necessary to reload the iSBC 957 Loader program or to download the program code a second time to the iSBC 86/12 board. It is only necessary to re-establish the communications link by typing two "U"s for the baud rate search.
2. The iSBC 86/12 board should not be plugged into an available card slot in an Inteltec chassis; a separate chassis should be used. There are at least three reasons for this:
 - a. There is only one RESET signal available on the Inteltec system bus. Thus, each processor may not be reset independently. This means that the iSBC 86/12 board cannot be reset without re-booting the ISIS-II operating system and restarting the iSBC 957 Loader.
 - b. The Inteltec system uses five of the eight available interrupts on the system bus. This severely restricts the range of interrupts available to the iSBC 86/12 board. Also, the iSBC 86/12 board cannot turn-off the interrupt lamps on the Inteltec front panel.
 - c. The iSBC 86/12 board may address up to 1 Megabyte of memory using a 20 bit address. Many Inteltec systems contain boards which generate and decode only the low order 16 address bits. For example, the iSBC 016 memory expansion board and the Inteltec 800

monitor PROMs only decode 16 address bits. Memory expansion above 64K bytes in these systems is difficult since the boards which decode only 16 bits will force "holes" in the address space above 64K.

3. The iSBC 86/12 board is delivered with two inputs to the 8259A Priority Interrupt Controller connected. Interrupt request 2 (IR2) is connected to the counter 0 output of the 8253 Programmable Interval Timer. IR5 is connected to the INT5/signal of the MULTIBUS System Bus. If these interrupts are not desired, the wire wrap jumpers making the connections should be removed from the iSBC 86/12 board. A particular problem may exist with the counter 0 connection to IR2. If the 8253 counter 0 is not specifically initialized with software, a low frequency square wave output will exist at counter 0's output. This may cause unwanted interrupts when interrupts are enabled by user programs.

4. If the iSBC 86/12 board is used in a system with expansion boards, it is important that the MULTIBUS bus exchange pins be properly jumpered. For example, if the iSBC 86/12 board is used with an iSBC 032 expansion memory board in a system, the BPRN/ MULTIBUS pin for the iSBC 86/12 board should be grounded.

In addition, if any interrupts are used with the iSBC 86/12 board the BPRN/ pin must be grounded. This is true in both single and multiple board systems.

5. Certain user systems require more than one single board computer in the system for performing the functions required by the application. The MULTIBUS System Bus has been specifically designed to permit multiple CPU boards to communicate and to share system resources. However, debugging systems with multiple CPUs has always posed somewhat of a problem. The iSBC 957 package provides a solution to this problem. The serial cable which connects the iSBC 86/12 board to the Inteltec system may be removed after the program has been downloaded to the iSBC 86/12 board. A console CRT may then be connected directly to the iSBC 86/12 board and the monitor program may be used to debug the program running on the board. Other iSBC 86/12 boards may also be downloaded from the Inteltec system and then switched to their own local terminals. An 8-bit processor board, such as the iSBC 80/30 board, may also be included

in the system and ICE-85™ may be used for debugging the iSBC 80/30 program concurrently with the iSBC 86/12 programs. Using this scheme, it is possible to debug a system which has several CPU boards by setting breakpoints and using other debugging features on each of the individual CPUs.

V. MATRIX MULTIPLICATION EXAMPLE

To illustrate how the iSBC 957 package can be used to assist in the writing and debugging of 8086 programs on the iSBC 86/12 board, an example program of a matrix multiplication will be presented. The example chosen has been intentionally kept simple and straightforward. The emphasis of this section will be to document the steps required to assemble, compile, link, locate and debug software using an Intellec system, the iSBC 957 package and the iSBC 86/12 board. Part of the example will be written in 8086 assembly language and part in PL/M-86.

The main program is written in PL/M-86. The main program first performs some initialization and the matrix multiplication, then the program calls an assembly language procedure (subroutine), a PL/M-86 procedure and the console output procedure CO supplied in the I/O library on the iSBC 957 diskette. A flow diagram for the example program is shown in Figure 6.

Explanation of the Program Code

The program code is contained in three software modules EXECUTION\$VEHICLE, FIND, and SBCCO. EXECUTION\$VEHICLE contains the main program coded in PL/M-86 and the binary to ASCII conversion procedure BIN\$DEC\$ASC also coded in PL/M-86. The module FIND contains the assembly language procedure FIND\$MX which searches a matrix for its maximum value. The module SBCCO resides in the library of console I/O routines supplied with the iSBC 957 package. The procedure CO will be used from this library.

The program code for the EXECUTION\$VEHICLE and FIND modules will be explained in the following paragraphs. Appendix B contains compilation and assembly listings for the two modules; also contained in Appendix B is a memory and debug map for the linked modules. The listings contain circled reference letters (e.g., (A)) which are referred to by the code description below. The listings in the appendix have been printed on fold-out pages so that they may easily be seen when reading the text.

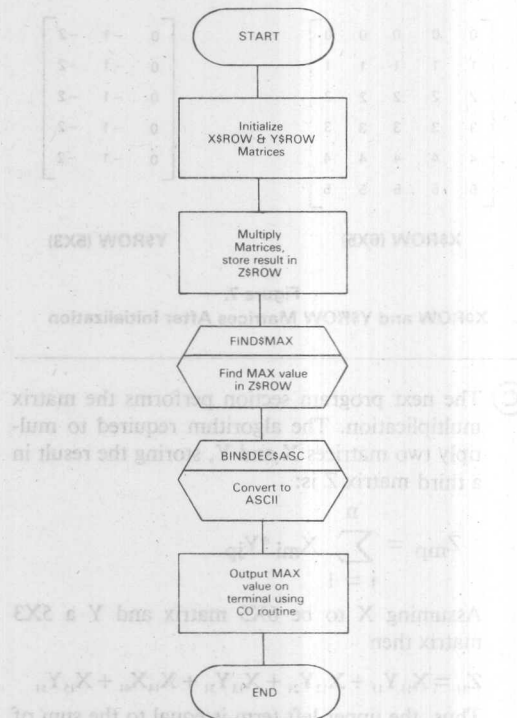


Figure 6.
Flow Diagram of Matrix Multiplication Example

Much of the description given below assumes that the reader is familiar with the PL/M-86 language and compiler, the 8086 assembler, and the link and locate program QRL86. It is recommended that the reader have at least a cursory knowledge of these subjects. The Intel literature for these subjects is listed near the front of this application note.

The EXECUTION\$VEHICLE Module

- (A) The first section of the module includes introductory comments and then statements to declare the matrices, other variables, and procedures used in the program. Note that the matrix dimensions are declared using the literals M, N, and P which are initially set to 6, 5, and 3. Later in this note, other values for M, N, and P will be used.
- (B) The next section of code contains the statements which initialize the two matrices that will be multiplied X\$ROW and Y\$ROW.

As a result of this initialization, the two matrices will contain values as shown in Figure 7.

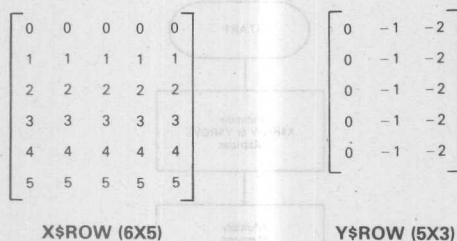


Figure 7.
X\$ROW and Y\$ROW Matrices After Initialization

- (C) The next program section performs the matrix multiplication. The algorithm required to multiply two matrices X and Y, storing the result in a third matrix Z is:

$$Z_{mp} = \sum_{i=1}^n X_{mi} * Y_{ip}$$

Assuming X to be 6X5 matrix and Y a 5X3 matrix then

$$Z_{11} = X_{11}Y_{11} + X_{12}Y_{21} + X_{13}Y_{31} + X_{14}Y_{41} + X_{15}Y_{51}$$

Thus, the upper left term is equal to the sum of the products of the top row of the X matrix times the left column of the Y matrix. The result that is obtained by multiplying the two matrices X\$ROW and Y\$ROW after they are initialized as explained above, is shown in Figure 8.

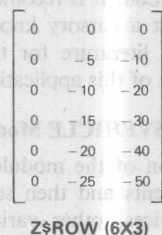


Figure 8. Result of Multiplying the Initialized Matrices X\$ROW and Y\$ROW

- (D) The external assembly language procedure FIND\$MX is called to determine the maximum value in the matrix. The procedure is a typed procedure and returns the maximum value to the calling program which stores it in the integer variable MAX.

- (E) The maximum value is then converted to a six (6) digit ASCII character string by the procedure BIN\$DEC\$ASC. The character string is stored in the array MAX\$ASC\$ARRAY, which contains the sign of the number and five (5) digits for the magnitude.
- (F) Finally, the characters "MAX VALUE =" are output on the system console followed by the 6 ASCII characters containing the maximum value. The PL/M-86 built-in procedure SIZE returns the number of bytes of the array TEXT as a word value. The PL/M-86 built-in procedure SIGNED changes the type of the value from WORD to INTEGER. This is required so that the type of the arguments in the DO statement agree. The console output procedure CO is used to output the characters on the system console.

- (G) Also contained in the module MATRIX.PLM is the binary to ASCII conversion procedure BIN\$DEC\$ASC. The first portion of the code contains the comments explaining the parameters and the calling sequence followed by the declarations. Note that the address of the array where the characters are to be stored is passed to the procedure and that the characters will be stored in the array using based variables. The next section of the code stores either a + or - sign in the first character position of the ASCII array and stores the absolute value of VALUE in the variable TEMP. Finally, the binary value is converted to ASCII using the algorithm explained in the comments. The MOD operator returns the remainder of the division by 10. The UNSIGN built-in procedure is required to change the type of the expression from INTEGER to WORD.

The FIND Module

- (H) The FIND module contains the assembly language procedure FINDMX. The calling sequence and the parameters are explained in the comments at the beginning of the listing. Note that the label FINDMX has been declared PUBLIC so the link program can fill in its address in the CALL statement in the main program of module EXECUTION\$VEHICLE.
- (I) The FIND module will contain three segments: a data segment, a stack segment and a code segment. It will be both convenient and pragmatic to append these three segments to the code, data and stack segments created by the

compiler for the EXECUTION\$VEHICLE module. To accomplish this, the three segments must be given the same SEGMENT and CLASS names as those given these segments by the compiler. The SEGMENT and CLASS names used by the compiler are CODE, DATA, and STACK. The GROUP statements are used to place the segments DATA and STACK in the group DGROUP and the segment CODE in the group CGROUP. These group definitions conform with the group definitions generated by the PL/M-86 compiler when the SMALL size control option is used. A group is a collection of segments which requires less than 64K bytes of memory.

The ASSUME directive informs the assembler that the DS and SS registers will contain the base address of DGROUP and the CS register will contain the base address of CGROUP. This information will be used by the assembler when constructing machine instructions.

- (J) The first segment appearing in the module is the data segment. The order of the segments is arbitrary, although it is recommended that the data segment precede the code segment to minimize forward references to variables which may cause the assembler to generate longer instruction codes. The data segment is declared PUBLIC, aligned on a WORD boundary and given both a segment and class name of DATA. Then follows the contents of the segment. In this particular example, only one word of storage is required. The ENDS directive indicates the end of the segment.

- (K) Next comes the stack segment which is given the segment name of STACK, the combine-type attribute of STACK and the class name of STACK. The combine-type attribute of STACK assures that the stack storage required in this module will be appended to the storage required in the PL/M-86 compiled modules. Two bytes of stack are required by the code in this module, however, the monitor uses 13 words of stack when breakpoints and interrupts are used. Therefore, 14 words are reserved for the stack.

- (L) Finally comes the code segment. The code segment has been given a segment name and class name of CODE and a group name of CGROUP, and has been declared PUBLIC. The alignment attribute of BYTE is specified

since it is desired that the code from this module be appended directly to the code from other modules without gaps between the code modules.

The assembly language code follows next. The code for the procedure must be enclosed between a pair of PROC, ENDP statements. The PROC statement is given the label FINDMX and specified as a NEAR procedure indicating it will be called with a near (intra-segment) CALL instruction and not a far (inter-segment) CALL instruction.

The comments at the beginning of the module and adjacent to the program statements explain the function being performed by the assembly language code.

The SBCCO Module

- (M) The console output procedure CO is contained in the object module SBCCO of the library file SBCIOS.LIB. SBCIOS.LIB is part of the iSBC 957 package I/O libraries. The calling sequence and parameters for CO may be seen in the external procedure declaration in the EXECUTION\$VEHICLE module.

Compiling the EXECUTION\$VEHICLE Module

The EXECUTION\$VEHICLE module is stored on a file named MATRIX.PLM on disk device :F1:. To compile the module, the following command line is used:

- PLM86 :F1:MATRIX.PLM DEBUG

This command line will cause the module stored in the file :F1:MATRIX.PLM to be compiled. The object code generated will be stored in a file with the default name :F1:MATRIX.OBJ and the listing generated will be stored in a file with the default name :F1:MATRIX.LST. To override the default object and listing files, the NOOBJECT and NOLIST compiler control switches can be used. File names for the listing and object files may also be specified in the command line. The DEBUG compiler control switch causes the compiler to generate extra symbol and line number information which will be used during debugging of the program. A listing of the compiled EXECUTION\$VEHICLE module is contained in Appendix B.

To aid in the debugging of the program, the module was compiled a second time with the following command line:

- PLM86 :F1:MATRIX.PLM NOOBJECT
CODE DEBUG PRINT (:F1:MATRIX.XLS)

This command line specified that no object file is to be created and a listing file should be stored in the file :F1:MATRIX.XLS. The CODE compiler control switch causes the compiler to list the assembly language statements which the compiler has generated for each line of PL/M code. The listing stored in the file MATRIX.XLS is contained in Appendix C.

Assembly of the FIND Module

The assembly language module FIND is stored on a file named FIND.ASM, to assemble this module the following command line is used:

ASM86 :F1:FIND.ASM DEBUG

This command line will cause the FIND module to be assembled with the object code stored in the default file :F1:FIND.OBJ and the listing stored in the default file :F1:FIND.LST. The listing of the assembled FIND module is contained in Appendix B.

Linking and Locating the Object Module

To link and locate the object modules, the QRL86 program will be used. The QRL86 program performs both the linking and the locating of the object modules in a single step. QRL86 is primarily designed for the debugging stages of program development. Some applications may require the extended capabilities of the separate LINK and LOCATE programs when the final link and locate is performed. The command line used to invoke the QRL86 program is:

QRL86 :F1:MATRIX.OBJ, :F1:FIND.OBJ,
SBCIOS.LIB ORIGIN (1000H)

This command line will cause QRL86 to link the code from the three modules and to locate the resultant absolute object module starting at location 1000 hexadecimal. The iSBC 86/12 monitor uses the first 180H bytes of memory for the monitor stack, data and interrupt vectors, 1000H was chosen as a convenient starting address for the program. The absolute object code will be stored in a default file :F1:MATRIX (note no file name extension is used). By default, the memory and debug maps which are generated are stored in the file :F1:MATRIX.MPQ and are contained in Appendix B.

(N) The memory map contains the starting addresses and sizes of the CODE, CONST, DATA, STACK and MEMORY segments of the object module. Note that the start address

for the program is specified as (0100H, 0002H) indicating a CS value of 0100H and an IP value of 0002H or an absolute value of 01002H. The first two bytes of the code segment contain address values which the code generated by the compiler will use for setting up the DS and SS registers. The memory map shows the code segments from the three modules collected into the group CGROUP. The code segment from the EXECUTION\$VEHICLE module is given the segment and class names of CODE and is put into CGROUP by the PL/M compiler. To assure that the code segment from the FIND module is concatenated with the code segment from the EXECUTION\$VEHICLE module the identical class, segment and group names were specified in the SEGMENT and GROUP statements in the FIND module. Next, the group DGROUP is shown in the memory map. DGROUP contains 4 segments labelled CONST, DATA, STACK and MEMORY. Putting all of these segments in the same group tells the linker that they will all be in the same 64K block of memory. The SMALL size control option of the compiler, which was invoked by default, creates CGROUP, DGROUP, and the segments contained in them.

(P) The debug map contains the memory address of variables, instruction labels and the addresses of each code line of the PL/M-86 module. Notice that the variable storage labels have their addresses specified in the format (DS register value, displacement). For example, the variable TEMP has an address of DS=012AH, displacement = 000CH or an absolute address of 0136H. Instruction labels and line numbers use the format (CS register value, IP register value). Thus, line number six (6) in the module EXECUTION\$VEHICLE has the address CS=0100H, IP=0B5H or 011B5H.

Object to Hex Conversion

Before downloading the program to the iSBC 86/12, the format of the object module must be converted from the absolute object module format which QRL86 creates to a hexadecimal/ASCII representation of the object module. This is done using the program OH86 with the following command line:

OH86 :F1:MATRIX TO :F1:MATRIX.HEX

Downloading and Debugging the Program

The hardware configuration used for debugging the matrix multiplication example program code was

an Intellec Series II Model 230 development system, the iSBC 957 package, an iSBC 86/12 board, and an iSBC 660 system chassis. What follows is the system-user dialog for a typical debugging session.

The first step required is to bootstrap load the ISIS-II operating system by hitting the RESET switch of the Intellec. The Intellec resident loader software is then loaded and executed. Throughout the dialog which follows operator entered characters will be underlined:

```
ISIS-II, V3.4
-SBC861
```

```
ISIS-II iSBC 86/12 LOADER, V1.2
```

To initialize the iSBC 86/12 monitor, the user must hit the RESET switch on the iSBC 660 chassis and type two "U"s on the system console. The monitor program will output a line on the console when it is properly initialized.

```
iSBC 86/12 MONITOR, V1.2
```

The monitor command "X" is typed to check that the monitor is properly operating and to examine the contents of the 8086 registers.

```
.X
AX=0000 BX=0000 CX=0000 DX=0000 SP=01C0 BP=0000 SI=0000
DI=0000 CS=0000 DS=0000 SS=0000 ES=0000 IP=0000 FL=0000
```

To download the hex object file to the iSBC 86/12, the "L" command is used. Because an Intellec Series II Model 230 is being used, a serial download is specified. The hex file name is MATRIX.HEX which is resident on disk device F1:.

```
.LS,:F1:MATRIX.HEX
```

The "X" command is used again to examine the CPU registers. Note that the monitor has changed the contents of the CS and IP registers to the value of the starting address of the program.

```
.X
AX=0000 BX=0000 CX=0000 DX=0000 SP=01C0 BP=0000 SI=0000
DI=0000 CS=0100 DS=0000 SS=0000 ES=0000 IP=0002 FL=0000
```

The "D" command is next used to display the first 101 bytes of the program code. Unless another segment register is specified, the display command assumes all addresses specified are relative to the CS register. Thus, the code displayed will be from absolute addresses 1000 through 1100. The program code displayed may be compared with program code generated by the PL/M-86 compiler shown in Appendix C, code line 36.

```

.D0 100
0000 2A 01 FA 3E 8E 16 00 00 0C D0 00 8B EC 16 1F FB
0010 C7 36 8E 00 00 00 81 3E 8E 00 05 00 7E 03 E9 3C
0020 00 C7 06 90 00 00 00 81 3E 90 00 04 00 7E 03 E9
0030 22 00 8B 06 8E 00 00 B9 0A 00 F7 E9 8B 36 90 00 D1
0040 E6 89 C3 8B 0E 8E 00 89 88 10 00 81 06 90 00 01
0050 00 E9 D3 FF 81 06 8E 00 01 00 E9 89 FF C7 06 8E
0060 00 00 00 81 3E 8E 00 04 00 7E 03 E9 40 00 C7 06
0070 90 00 00 00 81 3E 90 00 02 00 7E 03 E9 26 00 8B
0080 06 90 00 F7 D8 50 85 05 3E 00 B9 06 00 F7 E9 8B
0090 36 90 00 D1 E6 89 C3 59 89 88 4C 00 81 06 90 00
00A0 01 00 E9 CF FF 81 06 8E 00 01 00 E9 B5 FF C7 06
00B0 92 00 00 00 81 3E 92 00 02 00 7E 03 E9 8C 00 C7
00C0 06 8E 00 00 00 81 3E 8E 00 05 00 7E 03 E9 72 00
00D0 8B 06 8E 00 B9 06 00 F7 E9 8B 36 92 00 D1 E6 89
00E0 C3 C7 80 6A 00 00 00 C7 06 90 00 00 81 3E 90
00F0 00 04 00 7E 03 E9 41 00 8B 06 8E 00 B9 0A 00 F7
0100 E9

```

The PL/M-86 compiler ends the main program in the EXECUTION\$VEHICLE module with a halt instruction. After execution of the program it is more desirable to return to the monitor. To accomplish this, an INT 3 instruction (code=CC) will be substituted for the halt instruction (code=F4) at the address of 1B4H relative to a CS value of 100H. First the "D" command is used to verify the address of the halt instruction, then the "S" command is used to change the instruction to an INT 3 instruction.

```
.D1B4
01B4 F4
.S1B4, F4- CC
```

To execute the PL/M-86 main program, the "G" command is used. After the "G" is typed, the current contents of the IP are output, followed by the contents of the byte pointed to by the IP. A new value for the IP or breakpoint addresses may be specified before a carriage return <CR> is typed. In this example, only a <CR> is typed.

```
.G 0002- FA
MAX VALUE = -00050
00100:01B5 55
```

The program executes and outputs the maximum value of the matrix calculated. The INT 3 instruction is executed which causes a return to the monitor. The monitor types out an at-sign (@) followed by the CS and IP register values and the first byte of the instruction following the INT 3 instruction.

The "X" command is typed to examine the CPU registers. Note that the program has set both the SS and DS registers to 012A. (012A0H is the address of the DGROU as shown in the memory map.)

```
.X
AX=0030 BX=0005 CX=000A DX=0000 SP=0000 BP=0000 SI=0001
DI=0006 CS=0100 DS=012A SS=012A ES=0000 IP=01B5 FL=F202
```

The three matrices are displayed. Note that a word

display has been specified by using the "DW" Command and that the addresses have been specified relative to the DS register. The addresses of X\$ROW, Y\$ROW, and Z\$ROW may be found in the debug map given by QRL86. Note that the values stored in the matrices are the same as those shown in Figures 8 and 9.

```
.DW DS:10,4A
0010 0000 0000 0000 0000 0000 0001 0001 0001
0020 0001 0001 0002 0002 0002 0002 0002 0003
0030 0003 0003 0003 0003 0004 0004 0004 0004
0040 0004 0005 0005 0005 0005 0005 0005 0005
.DW DS:4C,68
004C 0000 FFFF
0050 FFFE 0000 FFFF FFFE 0000 FFFF FFFE 0000
0060 FFFF FFFE 0000 FFFF FFFE
.DW DS:6A,8C
006A 0000 0000 0000
0070 0000 FFFB FFF6 0000 FFF6 FFEC 0000 FFF1
0080 FFE2 0000 FFEC FFD8 0000 FFE7 FFCE
```

The "G" Command is used to reset the IP register to the start address of the program (002) and to specify a breakpoint at address 0AEH, which is the address of statement 57 of the main program. Statement 57 is the point in the program after the X\$ROW and Y\$ROW matrices have been initialized, but before the matrix multiplication is performed. After the <CR> is typed, the program executes until the breakpoint is encountered. At this point, the monitor outputs a line specifying the number of the breakpoint, the CS and IP values and the first byte of the next instruction to be executed.

```
.G 01B5- 55 002,AE
BR1 00100:00AE C7
```

Next, the single-step capability is used with the "N" command to execute single instructions. At any time, CPU registers may be examined or changed. In this example, the "X" command is used. Execution of succeeding instructions is caused by typing a comma (.).

```
.N 00AE- C7 .
00B4- 81 .
00BA- 7E .
00BF- C7 .
.X
AX=0018 BX=0018 CX=FFFF DX=0000 SP=00D0 BP=00D0 SI=0004
DI=0006 CS=0100 DS=012A SS=012A ES=0000 IP=00BF PL=F293
.N 00BF- C7 .
00C5- 81 .
00CB- 7E .
```

The contents of the X\$ROW and Y\$ROW matrices are examined and changed with the "SW" (substitute word) command. If a comma (,) is typed after the contents of memory are displayed, then the contents are left unchanged and the next word of memory is displayed. If a value followed by a comma or <CR> is entered, then the contents are changed. If a <CR> is entered, the substitute

sequence is terminated.

```
.SW DS:1A, 0001- .
001C 0001- .
001E 0001- 10
.SW DS:5A, FFFF- .
005C FFFE- .
005E 0000- .
0060 FFFF- 64
```

After the matrices are modified, execution is resumed with the "G" command. The max value is output and the INT 3 instruction executed. Finally, the contents of the 3 matrices are displayed.

```
.G 00CB- 7E
MAX VALUE = +00400
@0100:01B5 55
.DW DS:10,8C
0010 0000 0000 0000 0000 0000 0001 0001 0010
0020 0001 0001 0002 0002 0002 0002 0002 0003
0030 0003 0003 0003 0003 0004 0004 0004 0004
0040 0004 0005 0005 0005 0005 0005 0005 0005
0050 FFFE 0000 FFFF FFFE 0000 FFFF FFFE 0000
0060 0064 FFFE 0000 FFFF FFFE 0000 0000 0000
0070 0000 0051 FFD8 0000 00C0 FFEC 0000 0120
0080 FFE2 0000 0180 FFD8 0000 01E0 FFCE
```

Expanding the Example Program's Memory Requirements

To illustrate how the iSBC 86/12 board may be used for executing 8086 programs which require large amounts of RAM, the example program will be modified. The matrix dimensions of the example will be changed from values of 6, 5 and 3 for the literal symbols of M, N, and P to values of 100, 50, 70. The three matrices will then be of size 100X50, 50X70, and 100X70. The memory required for these matrices is 15.5K words or 31K bytes. The data, constant, stack and memory segments which are contained in the group DGROUP will now comprise almost 32K bytes of memory.

The extra memory requirements will be supplied by using an iSBC 032 board with the iSBC 86/12 board in the iSBC 660 chassis. The iSBC 032 board is a 32K byte RAM board which is compatible with both 8- and 16-bit CPU boards. The base address of the board may be selected anywhere in a 0 to 1 megabyte range on any 16K byte boundary. 8- or 16-bit data transfers may be selected. The iSBC 032 board will be jumpered to respond to addresses in the 512K or 544K address space (20 bit hex address range to 80000H to 87FFFH). This will illustrate the capabilities of the 8086 to access a 20-bit, 1 megabyte address range.

One other modification is required to the program. The magnitude of the numbers which would result from multiplying matrices of this size would greatly exceed the capacity of the 16-bit integer storage, even with the two matrices initialized to the small

values they presently contain. To keep the example simple, the initialization values will be changed so all elements of the X\$ROW matrix are set equal to 2 and all elements of the Y\$ROW matrix are set equal to 3. The result of the multiplication should make all the elements of Z\$ROW equal to 300.

The modified lines of program code are shown below.

```

27 1      /* MATRIX DIMENSIONS */
28 1      DECLARE M LITERALLY '100';
29 1      DECLARE N LITERALLY '50';
30 1      DECLARE P LITERALLY '70';

36 1      DO I = 0 TO (M-1);
37 2          DO J = 0 TO (N-1);
38 3              X$ROW(I).COL(J) = 2;
39 3          END;
40 2      END;

41 1      DO I = 0 TO (N-1);
42 2          DO J = 0 TO (P-1);
43 3              Y$ROW(I).COL(J) = 3;
44 3          END;
45 2      END;

```

The EXECUTION\$VEHICLE module must be re-compiled and then the three program modules must be linked and located using the QRL86 program. Specifying the SEGMENTS option of QRL86, the origin of the CODE segment which is in the group CGROUP is set at 1000H, as in the first example. However, the origin of the CONST, DATA STACK and MEMORY segments which make up the group DGROUP is set at 8000H.

QRL86 :F1:MATRIY.OBJ,:F1:FIND.OBJ,
SBCIOS.LIB SEGMENTS (CODE(1000H),
CONST (8000H), DATA STACK, MEMORY)

The memory map generated by QRL86 shows the CGROUP having a start address of 01000H and the DGROUP having a start address of 8000H.

INVOKED BY:
QRL86 :F1:MATRIY.OBJ,:F1:FIND.OBJ,SBCIOS.LIB &
SEGMENTS (CODE(1000H),CONST (8000H),DATA,STACK,MEMORY)

INPUT MODULES INCLUDED:
:F1:MATRIY.OBJ (EXECUTIONVEHICLE)
:F1:FIND.OBJ (FIND)
SBCIOS.LIB (SBCCO)

RESULT WRITTEN TO :F1:MATRIY (EXECUTIONVEHICLE)
START ADDRESS IS (0100H,0002H)

START	LTH	ALIGN	NAME	CLASS
01000H	290H	G	/GS/ CGROUP	
01000H	210H	W	CODE (EXECUTIONVEHICLE)	CODE
01210H	41H	B	CODE (FIND)	CODE
0125EH	3AH	W	CODE (SBCCO)	CODE
80000H	7970H	G	/GE/ DGROUP	
80000H	CH	W	CONST (EXECUTIONVEHICLE)	CONST
8000CH	0H	W	CONST (SBCCO)	CONST
8000CH	792AH	W	DATA (EXECUTIONVEHICLE)	DATA
87936H	2H	W	DATA (FIND)	DATA
87938H	0H	W	DATA (SBCCO)	DATA
87940H	30H	SW	STACK	STACK
87970H	0H	W	MEMORY	MEMORY
87970H	0H	G	/GE/ DGROUP	
87970H	0H	G	??SEG (FIND)	(NULL)

The object code is then converted to hex format and downloaded to the iSBC 86/12 board. When the program is executed, the maximum value is calculated and output on the console.

```

-SBC861
ISIS-II iSBC 86/12 LOADER, V1.2

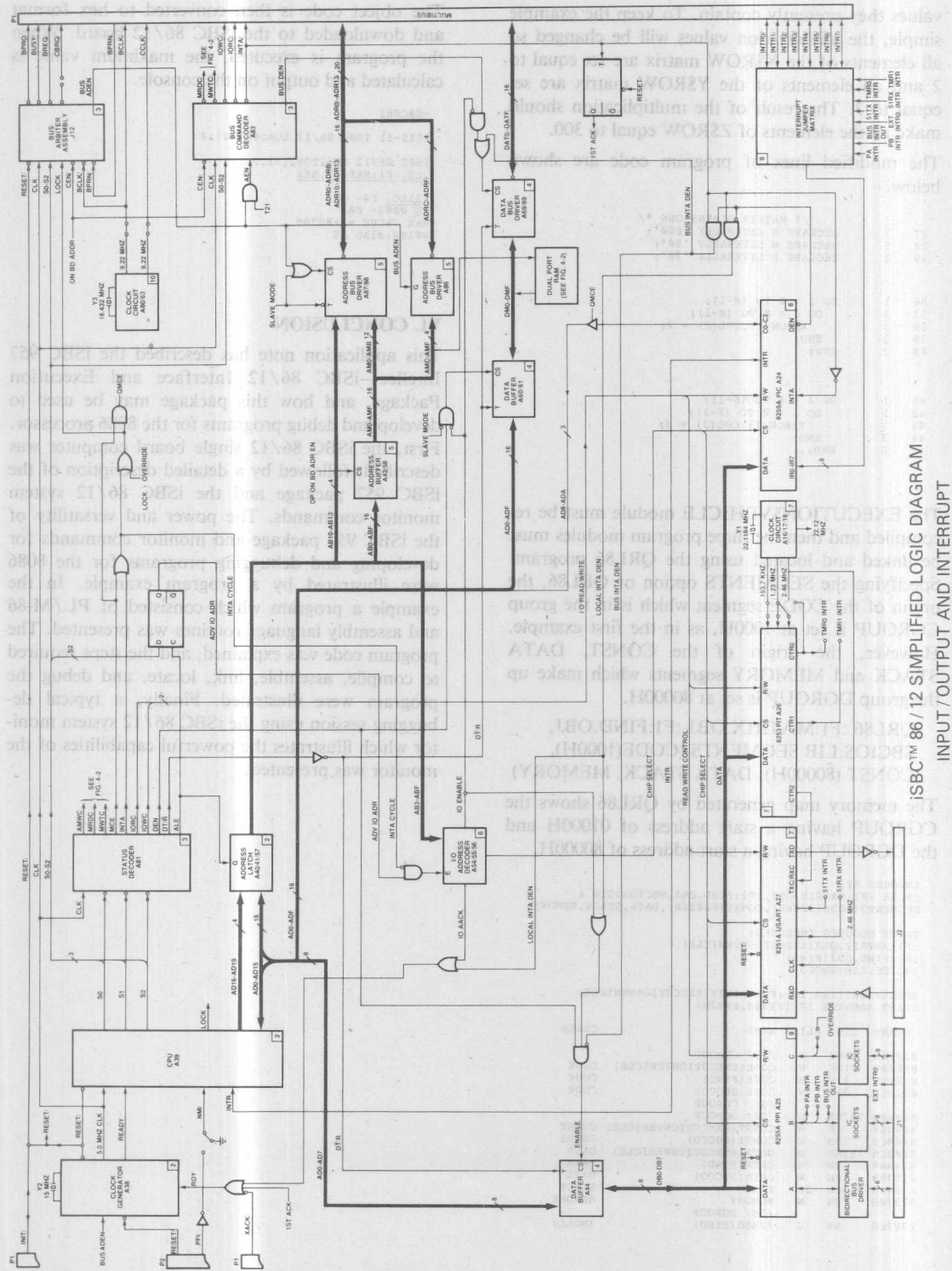
iSBC 86/12 MONITOR, V1.2
.LS,:F1:MATRIY.HEX

.SIAC, F4- CC
.G 0002- FA
MAX VALUE = +00300
00100:01AD 55

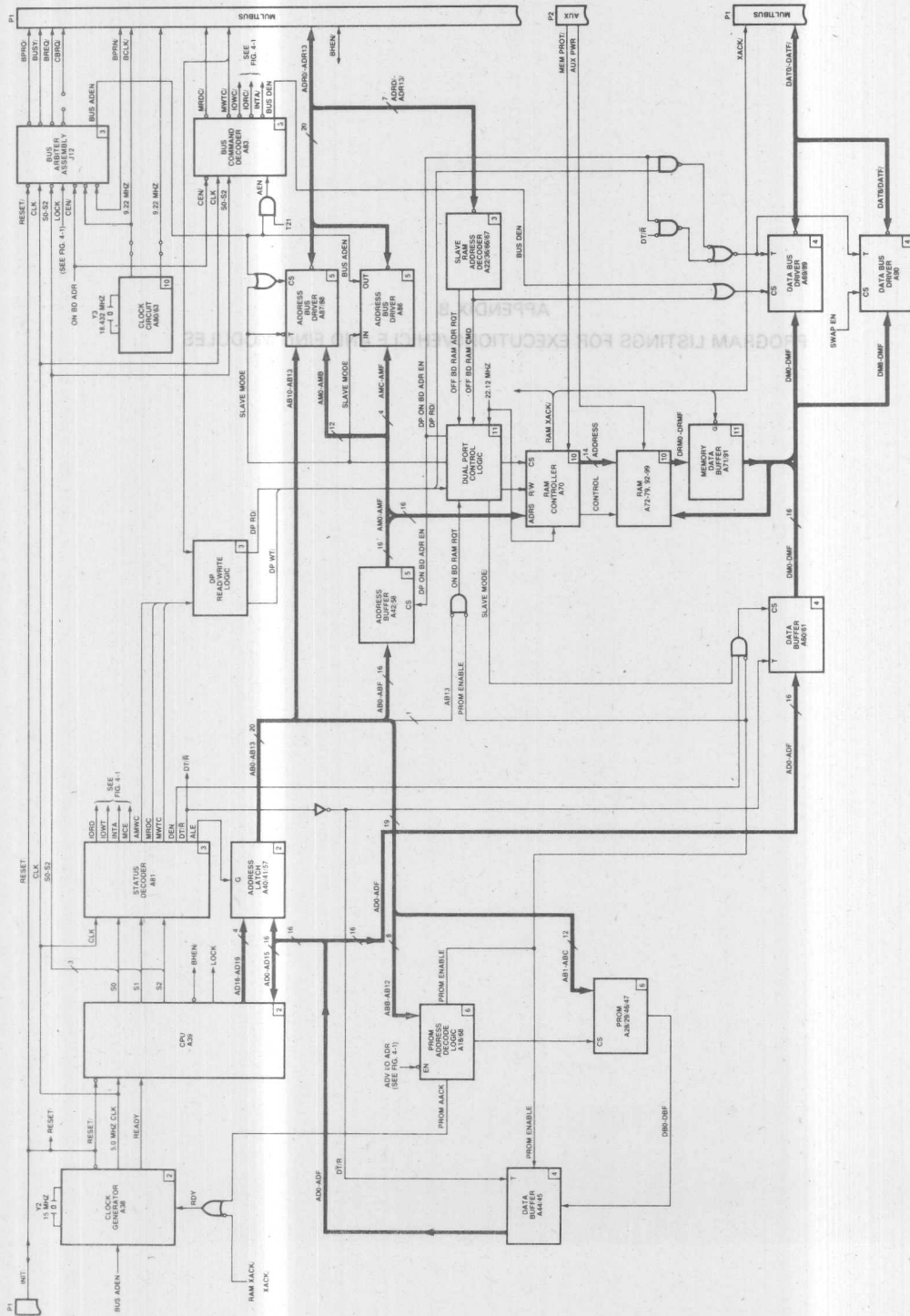
```

VI. CONCLUSION

This application note has described the iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package, and how this package may be used to develop and debug programs for the 8086 processor. First, the iSBC 86/12 single board computer was described, followed by a detailed description of the iSBC 957 package and the iSBC 86/12 system monitor commands. The power and versatility of the iSBC 957 package and monitor commands for developing and debugging programs for the 8086 were illustrated by a program example. In the example a program which consisted of PL/M-86 and assembly language routines was presented. The program code was explained, and the steps required to compile, assemble, link, locate, and debug the program were illustrated. Finally, a typical debugging session using the iSBC 86/12 system monitor which illustrates the powerful capabilities of the monitor was presented.



APPENDIX A (2 of 2)

iSBC™ 86/12 SIMPLIFIED LOGIC DIAGRAM
ROM/EPROM AND DUAL PORT RAM

APPENDIX B

PROGRAM LISTINGS FOR EXECUTION\$VEHICLE AND FIND MODULES

MANOALD DIOIC SIMPLIFIED LOGIC
MAR 1904-1AUG DIA MORP/1MOR
1280-88 12 SIMPLIFIED LOGIC

/* MATRIX MULTIPLICATION EXAMPLE PROGRAM

PL/M-86 MAIN PROGRAM WHICH:

- A) INITIALIZES TWO INTEGER MATRICES
- B) MULTIPLIES THE TWO MATRICES AND STORES THE RESULT IN A THIRD MATRIX
- C) CALLS AN ASSEMBLY LANGUAGE PROCEDURE WHICH SEARCHES THE THIRD MATRIX FOR THE MAXIMUM VALUE
- D) CALLS A PL/M PROCEDURE WHICH CONVERTS THE MAXIMUM VALUE FROM INTEGER TO ASCII
- E) CALLS A PROCEDURE WHICH OUTPUTS THE ASCII CHARACTERS ON THE SYSTEM CONSOLE

/*

1 EXECUTIONSVEHICLE;
 DO;

/* FINDSMX - EXTERNAL ASSEMBLY LANGUAGE PROCEDURE WHICH SEARCHES A MATRIX FOR THE LARGEST ABSOLUTE MAGNITUDE.

PARAMETERS:

MATRIX\$ADR - ADDRESS OF THE MATRIX TO BE SEARCHED
 ROWS - NUMBER OF ROWS IN THE MATRIX
 COLS - NUMBER OF COLUMNS IN THE MATRIX

/*

2 1 FINDSMX: PROCEDURE (MATRIX\$PTR, ROWS, COLS) INTEGER EXTERNAL;
 3 2 DECLARE (ROWS, COLS) INTEGER;
 4 2 DECLARE MATRIX\$PTR POINTER;
 5 2 END FINDSMX;

/* BINSDECSASC - BINARY TO DECIMAL ASCII CONVERSION PROCEDURE

PARAMETERS:

VALUE - INTEGER VALUE TO BE CONVERTED TO ASCII
 CHARSARRAY\$ADR - ADDRESS OF 6 BYTE ARRAY WHERE ASCII STRING CONTAINING THE VALUE WILL BE STORED

/*

6 1 BINSDECSASC: PROCEDURE (VALUE, CHARSARRAY\$ADR);
 7 2 DECLARE (VALUE, TEMP, I) INTEGER;
 8 2 DECLARE CHARSARRAY\$ADR POINTER;
 9 2 DECLARE (CHAR\$ARRAY BASED CHARSARRAY\$ADR) (6) BYTE;
 10 2 IF VALUE < 0 THEN
 11 2 DO;
 12 2 CHARSARRAY(0) = '-'; /* SIGN CHARACTER */
 13 2 TEMP = -VALUE;
 14 2 END;
 15 2 DO;
 16 2 CHARSARRAY(0) = '+';
 17 2 TEMP = VALUE;
 18 2 END;
 19 2 DO I = 5 TO 1 BY -1;
 20 2 CHARSARRAY(I) = UNSIGN(TEMP MOD 10) + 30H;
 21 2 TEMP = TEMP/10;
 22 2 /* ASCII CHARACTERS 30 THRU 39 HEX REPRESENT THE DIGITS 0 THRU 9. THUS
 23 2 TO CONVERT AN INTEGER TO ASCII REPEATED DIVISIONS BY 10 AND ADDING
 24 2 THE REMAINDER TO 30 HEX WILL ACCOMPLISH THE CONVERSION */
 25 2 END;
 26 2 END BINSDECSASC;

/* CO - EXTERNAL PROCEDURE TO OUTPUT A CHARACTER TO THE SYSTEM CONSOLE.
 THIS PROCEDURE IS PART OF THE ISBC 957 LIBRARY FOR CONSOLE I/O

PARAMETER:

CHAR - ASCII CHARACTER TO BE OUTPUT ON THE CONSOLE

/*

24 1 CO: PROCEDURE (CHAR) EXTERNAL;
 25 2 DECLARE CHAR BYTE;
 26 2 END CO;

/* MATRIX DIMENSIONS */

27 1 DECLARE M LITERALLY '6';
 28 1 DECLARE N LITERALLY '5';
 29 1 DECLARE P LITERALLY '3';

/* THE THREE MATRICES ARE DECLARED AS ARRAYS OF STRUCTURES. XSROW IS COMPOSED OF M STRUCTURES EACH OF WHICH IS COMPOSED OF N INTEGER ELEMENTS. THUS XSROW MAY BE THOUGHT OF AS A M X N MATRIX. THE MATRIX WILL BE STORED AS A ROW-ORDER MATRIX WITH THE ELEMENTS OF EACH ROW STORED IN ADJACENT MEMORY LOCATIONS. Y\$ROW IS DECLARED AS A N X P MATRIX AND Z\$ROW AS A N X P MATRIX */

30 1 DECLARE XSROW(M) STRUCTURE (COL(N) INTEGER);
 31 1 DECLARE Y\$ROW(N) STRUCTURE (COL(P) INTEGER);
 32 1 DECLARE Z\$ROW(M) STRUCTURE (COL(P) INTEGER);
 33 1 DECLARE (I,J,K,MAX) INTEGER;
 34 1 DECLARE MAX\$ASC\$ARRAY(6) BYTE;
 35 1 DECLARE TEXT(*) BYTE DATA ('MAX VALUE = ');

```

/* INITIALIZE XSROW SUCH THAT THE FIRST ROW IS SET EQUAL TO 0, THE SECOND
ROW EQUAL TO 1, THE THIRD ROW EQUAL TO 2, ETC. */
36 1 DO I = 0 TO (M-1);
37 2 DO J = 0 TO (N-1);
38 3 XSROW(I).COL(J) = I;
39 3 END;
40 2 END;

/* INITIALIZE YSROW SUCH THAT THE FIRST COLUMN IS SET EQUAL TO 0, THE
SECOND COLUMN EQUAL TO -1, AND THE THIRD COLUMN EQUAL TO -2. */
41 1 DO I = 0 TO (N-1);
42 2 DO J = 0 TO (P-1);
43 3 YSROW(I).COL(J) = -J;
44 3 END;
45 2 END;

/* PERFORM MATRIX MULTIPLICATION */
46 1 DO K = 0 TO (P-1);
47 2 DO I = 0 TO (M-1);
48 3 ZSROW(I).COL(K) = 0; /* SET ZSROW ELEMENT TO 0 */
49 3 DO J = 0 TO (N-1); /* SUM THE PRODUCT OF XSROW ROW TERMS AND YSROW COLUMN TERMS */
50 4 ZSROW(I).COL(K) = ZSROW(I).COL(K) + (XSROW(I).COL(J) * YSROW(J).COL(K));
51 4 END;
52 3 END;
53 2 END;

54 1 MAX = FINDSMX (ZSROW, M, P); /* FIND MAX VALUE OF ZSROW */
55 1 CALL BINSDECSASC (MAX, @MAX$ASCSARRAY); /* CONVERT TO DECIMAL ASCII */
56 1 DO I = 0 TO (SIGNED(SIZE(TEXT)) - 1); /* OUTPUT HEADER TEXT */
57 2 CALL CO(TEXT(I));
58 2 END;
59 1 DO I = 0 TO 5; /* OUTPUT ASCII MAX VALUE */
60 2 CALL CO(MAX$ASCSARRAY(I));
61 2 END;

62 1 END EXECUTION$VEHICLE;

MODULE INFORMATION:
CODE AREA SIZE = 0225H 549D
CONSTANT AREA SIZE = 0000H 12D
VARIABLE AREA SIZE = 0090H 144D
MAXIMUM STACK SIZE = 0000H 0D
137 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION

ISIS-II MCS-86 ASSEMBLER ASSEMBLY OF MODULE FIND
OBJECT MODULE PLACED IN :F1:FIND.OBJ
ASSEMBLER INVOKED BY: ASM86 :F1:FIND.ASM DEBUG

LOC OBJ

LINE SOURCE
1 NAME FIND
2 PUBLIC FINDMX
3 ;
4 ;
5 ;
6 ;
7 ;
8 ; ASSEMBLY LANGUAGE PROCEDURE TO FIND THE ELEMENT OF AN INTEGER
9 ; MATRIX WITH THE LARGEST ABSOLUTE MAGNITUDE. THE VALUE OF THE
10 ; ELEMENT IS RETURNED IN THE AX REGISTER.
11 ;
12 ; PL/M CALLING SEQUENCE:
13 ; MAX$VALUE = FINDSMX(ADRSOF$MATRIX, #SOF$ROWS, #SOF$COLS);
14 ;
15 ; PARAMETERS:
16 ; ADRSOF$MATRIX - ADDRESS OF THE MATRIX WHICH WILL BE SEARCHED
17 ; #SOF$ROWS - NUMBER OF ROWS IN THE MATRIX
18 ; #SOF$COLS - NUMBER OF COLUMNS IN THE MATRIX
19 ;
20 ; PL/M WILL PASS THE THREE PARAMETERS IN THE CALL TO THIS PROCEDURE ON
21 ; THE STACK. ON ENTRY TO THE PROCEDURE SP+6 WILL POINT TO THE FIRST
22 ; PARAMETER(ADRSOF$MATRIX) AND SP+4 AND SP+2 WILL POINT TO THE SECOND
23 ; AND THIRD PARAMETERS.
24 ;
25 ; THE PROCEDURE IS A TYPED PROCEDURE WHICH ASSIGNS THE MAXIMUM VALUE
26 ; IN THE MATRIX TO A VARIABLE (IN THIS CASE MAX$VALUE) IN A PL/M
27 ; ASSIGNMENT STATEMENT. TO ACCOMPLISH THIS ASSIGNMENT THE VALUE IS
28 ; RETURNED IN THE AX REGISTER.
29 ;
30 ; THE ALGORITHM USED IS SIMILAR TO THE FOLLOWING PL/M CODE:
31 ; FOR I = 0 TO (#SOF$ROWS - 1);
32 ; FOR J = 0 TO (#SOF$COLS - 1);
33 ; IF IABS(MATRIX(I).Y(J)) > IABS(MAX) THEN MAX = MATRIX(I).Y(J);
34 ; END;
35 ;
36 ;
37 ; WHERE IABS(XYZ) REPRESENTS THE ABSOLUTE VALUE OF THE INTEGER XYZ
38 ;
39 ;

```

LOC	OBJ	LINE	SOURCE
		40	;
		41	;
		42	DEFINE GROUPS TO CONFORM WITH PL/M-86 CONVENTIONS. DATA, STACK, AND
		43	CODE SEGMENTS WILL BE APPENDED TO THEIR RESPECTIVE SEGMENTS IN THE
		44	PL/M-86 MODULES.
		45	DGROUP DATA,STACK
		46	CGROUP GROUP CODE
		47	;
		48	INSTRUCT THE ASSEMBLER THAT THE DS, SS, AND CS REGISTERS WILL CONTAIN
		49	THE BASE ADDRESS VALUES FOR THE DGROUP, DGROUP AND CGROUP GROUPS.
		50	ASSUME DS:DGROUP,SS:DGROUP,CS:CGROUP
		51	;
		52	;
		53	*****DATA SEGMENT
		54	;
		55	DATA SEGMENT WORD PUBLIC 'DATA'
		56	MAX DW 0
		57	DATA ENDS
		58	;
		59	*****STACK SEGMENT
		60	;
		61	STACK SEGMENT STACK 'STACK'
		62	DW 14 DUP (0) ;RESERVE 13 WORDS OF STACK FOR MONITOR
		63	;
		64	STACK ENDS ;AND 1 WORD FOR FINDMX PROCEDURE
		65	;
		66	*****CODE SEGMENT
		67	;
		68	CODE SEGMENT BYTE PUBLIC 'CODE'
		69	;
		70	PARAMETERS ON STACK, DISPLACEMENT FROM TOS INCREASED BY TWO DUE TO INITIAL PUSH
		71	NO OF ROWS EQU WORD PTR [BP+6]
		72	NO OF COLS EQU WORD PTR [BP+4]
		73	ADR_OF_MATRIX EQU WORD PTR [BP+8]
		74	;
		75	FINDMX PROC NEAR ;PROCEDURE DECLARATION
		76	PUSH BP ;SAVE BP REGISTER
		77	MOV BP,SP ;BP POINTS TO PARAMETERS ON STACK
		78	XOR DX,DX ;SET DX = ABS OF CURRENT MAX = 0
		79	MOV DI,DX ;DI = I (ROW INDEX) = 0
		80	MOV SI,DX ;SI = J (COLUMN INDEX) = 0
		81	MOV MAX,DX ;MAX = CURRENT MAX = 0
		82	MOV CX,NO OF COLS
		83	SHL CX,1 ;CX = (#OFSCOLS) * 2
		84	;
		85	MOV BX,ADR OF MATRIX ;ADRSOFSMATRIX PARAMETER
		86	;
		87	ABC: MOV AX,[BX][SI] ;BX POINTS TO FIRST ELEMENT OF A GIVEN ROW
		88	OR AX,AX ;GET ELEMENT OF MATRIX
		89	JNS DEF ;SET FLAGS
		90	NEG AX ;JUMP IF SIGN = 0
		91	DEF: CMP AX,DX ;NEGATE TO FORM POSITIVE NUMBER
		92	JL XYZ ;COMPARE TO CURRENT MAX
		93	MOV DX,AX ;JUMP IF LESS THAN CURRENT MAX
		94	MOV AX,[BX][SI] ;MOVE TO ABS OF CURRENT MAX
		95	MOV MAX,AX ;MOVE MATRIX VALUE TO CURRENT MAX
		96	XYZ: ADD SI,2 ;INCREMENT J INDEX BY TWO
		97	CMP SI,CX ;END OF THIS ROW ??
		98	JB ABC ;IF NO, LOOP BACK FOR NEXT ELEMENT OF THIS ROW
		99	LEA BX,[BX+SI] ;BX = BX + (2 * #OFSCOLS), BX POINTS TO NEXT ROW
		100	MOV SI,0 ;J = 0
		101	INC DI ;I = I + 1
		102	CMP DI,NO OF ROWS ;LAST ROW ??
		103	JB ABC ;IF NO, DO THE NEXT ROW
		104	MOV AX,MAX ;RETURN MAX VALUE IN AX REGISTER
		105	POP BP ;RESTORE BP REGISTER
		106	RET 6 ;INCREMENT SP BY 6 AND RETURN TO CALLER
		107	;
		108	FINDMX ENDP
		109	;
		110	CODE ENDS
		111	;
		112	END

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
??SEG	SEGMENT		SIZE=0000H PARA PUBLIC
ABC	L NEAR	0015H	CODE
ADR OF MATRIX	V WORD	0008H	[BP]
CGROUP	GROUP		CODE
CODE	SEGMENT		SIZE=0041H BYTE PUBLIC 'CODE'
DATA	SEGMENT		SIZE=002H WORD PUBLIC 'DATA'
DEF	L NEAR	001DH	CODE
DGROUP	GROUP		DATA STACK
FINDMX	L NEAR	0000H	CODE PUBLIC
MAX	V WORD	0000H	DATA
NO OF COLS	V WORD	0004H	[BP]
NO OF ROWS	V WORD	0006H	[BP]
STACK	SEGMENT		SIZE=001CH PARA STACK 'STACK'
XYZ	L NEAR	0028H	CODE

ASSEMBLY COMPLETE, NO ERRORS FOUND

ISIS-II QRL-P6, V1.1

INVOKED BY:
QRL84 : F1: MATRIX.OBJ, : F1: FIND.OBJ, SBCIOS.LIB ORIGIN(1000H)

INPUT MODULES INCLUDED:
: F1: MATRIX.OBJ (EXECUTIONVEHICLE)
: F1: FIND.OBJ (FIND)
: SBCIOS.LIB (SBCCO)

RESULT WRITTEN TO : F1: MATRIX (EXECUTIONVEHICLE)
START ADDRESS IS (0100H, 2002H)

START	LTH	ALIGN	NAME	CLASS
01000H	2A0H	G	/GS/ CGROUP	
01000H	225H	W	CODE (EXECUTIONVEHICLE)	CODE
01225H	41H	B	CODE (FIND)	CODE
01266H	3AH	W	CODE (SBCCO)	CODE
012A0H	D0H	G	/GS/ DGROUP	
012A0H	CH	W	CONST (EXECUTIONVEHICLE)	CONST
012ACH	0H	W	CONST (SBCCO)	CONST
012ACH	90H	W	DATA (EXECUTIONVEHICLE)	DATA
0133CH	2H	W	DATA (FIND)	DATA
0133CH	0H	W	DATA (SBCCO)	DATA
01340H	30H	SW	STACK	STACK
01370H	0H	W	MEMORY	MEMORY
01370H	0H	G	/GE/ DGROUP	
01370H	0H	G	??SEG (FIND)	(NULL)

DEBUG MAP OF : F1: MATRIX (EXECUTIONVEHICLE)

MODULE: EXECUTIONVEHICLE	0100H, 01E1H	LINE #: 19
012AH, 00D0H SYMBOL: MEMORY	0100H, 01FBH	LINE #: 20
0100H, 01B5H SYMBOL: BINDECASC	0100H, 0213H	LINE #: 21
012AH, 000CH SYMBOL: TEMP	0100H, 021EH	LINE #: 22
012AH, 000EH SYMBOL: I	0100H, 0221H	LINE #: 23
012AH, 0010H SYMBOL: XROW	0100H, 0002H	LINE #: 36
012AH, 0004H SYMBOL: YROW	0100H, 0021H	LINE #: 37
012AH, 006AH SYMBOL: ZROW	0100H, 0032H	LINE #: 38
012AH, 008EH SYMBOL: I	0100H, 004BH	LINE #: 39
012AH, 0090H SYMBOL: J	0100H, 0054H	LINE #: 40
012AH, 0092H SYMBOL: K	0100H, 005DH	LINE #: 41
012AH, 0094H SYMBOL: MAX	0100H, 006EH	LINE #: 42
012AH, 0096H SYMBOL: MAXASCARRAY	0100H, 007FH	LINE #: 43
012AH, 0000H SYMBOL: TEXT	0100H, 009CH	LINE #: 44
0100H, 01B5H LINE #: 6	0100H, 00A5H	LINE #: 45
0100H, 01B0H LINE #: 10	0100H, 00A2H	LINE #: 46
0100H, 01C2H LINE #: 12	0100H, 00BFH	LINE #: 47
0100H, 01C8H LINE #: 13	0100H, 00D0H	LINE #: 48
0100H, 01D1H LINE #: 14	0100H, 00E7H	LINE #: 49
0100H, 01D4H LINE #: 16	0100H, 00F8H	LINE #: 50
0100H, 01DAH LINE #: 17	0100H, 013CH	LINE #: 51

0100H, 0139H	LINE #: 52
0100H, 0142H	LINE #: 53
0100H, 014BH	LINE #: 54
0100H, 015EH	LINE #: 55
0100H, 0169H	LINE #: 56
0100H, 017AH	LINE #: 57
0100H, 0185H	LINE #: 58
0100H, 018EH	LINE #: 59
0100H, 019FH	LINE #: 60
0100H, 01AAH	LINE #: 61
0100H, 01B3H	LINE #: 62
MODULE: FIN	
0100H, 023AH	SYMBOL: ABC
0100H, 0242H	SYMBOL: DEF
0100H, 0225H	SYMBOL: FINDMX
012AH, 009CH	SYMBOL: MAX
0100H, 024DH	SYMBOL: XYZ
0100H, 0225H	PUBLIC: FINDMX
MODULE: SBCCO	
0100H, 0266H	PUBLIC: CO

APPENDIX C

PROGRAM LISTING FOR EXECUTION\$VEHICLE MODULE WITH CODE EXPANSION

PL/M-86 COMPILER EXECUTION\$VEHICLE

ISIS-II PL/M-86 V1.0 COMPILATION OF MODULE EXECUTION\$VEHICLE
NO OBJECT MODULE REQUESTED.

COMPILER INVOKED BY: PLM86 :F1:MATRIX.PLM DEBUG CODE NOOBJECT PRINT(:F1:MATRIX.XLS)

/* MATRIX MULTIPLICATION EXAMPLE PROGRAM

PL/M-86 MAIN PROGRAM WHICH:

- A) INITIALIZES TWO INTEGER MATRICES
- B) MULTIPLIES THE TWO MATRICES AND STORES THE RESULT IN A THIRD MATRIX
- C) CALLS AN ASSEMBLY LANGUAGE PROCEDURE WHICH SEARCHES THE THIRD MATRIX FOR THE MAXIMUM VALUE
- D) CALLS A PL/M PROCEDURE WHICH CONVERTS THE MAXIMUM VALUE FROM INTEGER TO ASCII
- E) CALLS A PROCEDURE WHICH OUTPUTS THE ASCII CHARACTERS ON THE SYSTEM CONSOLE

*/

1 EXECUTION\$VEHICLE:
DO;

/* FINDSMX - EXTERNAL ASSEMBLY LANGUAGE PROCEDURE WHICH SEARCHES A MATRIX FOR THE LARGEST ABSOLUTE MAGNITUDE.

PARAMETERS:

MATRIX\$ADR - ADDRESS OF THE MATRIX TO BE SEARCHED

ROWS - NUMBER OF ROWS IN THE MATRIX

COLS - NUMBER OF COLUMNS IN THE MATRIX

*/

2 1 FINDSMX: PROCEDURE (MATRIX\$PTR, ROWS, COLS) INTEGER EXTERNAL;
3 2 DECLARE (ROWS, COLS) INTEGER;
4 2 DECLARE MATRIX\$PTR POINTER;
5 2 END FINDSMX;

/* BINSDECSASC - BINARY TO DECIMAL ASCII CONVERSION PROCEDURE
PARAMETERS:

VALUE - INTEGER VALUE TO BE CONVERTED TO ASCII

CHAR\$ARRAY\$ADR - ADDRESS OF 6 BYTE ARRAY WHERE ASCII

STRING CONTAINING THE VALUE WILL BE STORED

*/

6 1 BINSDECSASC: PROCEDURE (VALUE, CHAR\$ARRAY\$ADR);
; STATEMENT # 5

BINDECSASC PROC NEAR
01B5 55 PUSH BP
01B6 8BEC MOV BP,SP

7 2 DECLARE (VALUE, TEMP, I) INTEGER;
8 2 DECLARE CHAR\$ARRAY\$ADR POINTER;
9 2 DECLARE (CHAR\$ARRAY BASED CHAR\$ARRAY\$ADR) (6) BYTE;

10 2 IF VALUE < 0 THEN

01B8 81E06000 CMP [BP].VALUE,0H ; STATEMENT # 10

01BD 7C03 JL \$+5H

01BF E91200 JMP @1

11 2 DO;

12 3 CHAR\$ARRAY(0) = '-'; /* SIGN CHARACTER */

01C2 8B5E04 MCY BX,[BP].CHAR\$ARRAY\$ADR ; STATEMENT # 12

01C5 C6072D MOV CHAR\$ARRAY[BX],2DH

13 3 TEMP = -VALUE;

01C8 8B4606 MOV AX,[BP].VALUE ; STATEMENT # 13

01CB F7D8 NEG AX

01CD 890600 MOV TEMP,AX

14 3

END;

01D1 E90D00 JMP @2 ; STATEMENT # 14

ELSE

01D4 8B5E04 MCY BX,[BP].CHAR\$ARRAY\$ADR ; STATEMENT # 15

01D7 C6072B MOV CHAR\$ARRAY[BX],7BH

17 3

TEMP = VALUE;

01DA 8B4606 MOV AX,[BP].VALUE ; STATEMENT # 17

01DD 890600 MCY TEMP,AX

18 3

END;

01E1 C70602000500 MOV I,5H ; STATEMENT # 19

01E7 E90600 JMP @3

01EA 81060200FFFF ADD I,0FFFFH

```

                                #5:
01F0 813E0200100    CMP    I,1H
01F6 7D03          JGE    $+5H
01F8 E92600      JMP    @4
20 3    CHARSARRAY(I) = UNSIGN(TEMP MOD 10) + 30H; ; STATEMENT # 20
01FB 80060000      MOV    AX,TEMP
01FF B90A00      MOV    CX,0AH
0202 31D2          XOR    DX,DX
0204 F7F9          IDIV   CX
0206 81C23000      ADD    DX,30H
020A 8B5E04      MOV    BX,[BP].CHARARRAYADR
020D 8B360200      MOV    SI,I
0211 8B10      MOV    [BX].CHARARRAY[SI],DL
21 3    TEMP = TEMP/10; ; STATEMENT # 21
/* ASCII CHARACTERS 30 THRU 39 HEX REPRESENT THE DIGITS 0 THRU 9. THUS
TO CONVERT AN INTEGER TO ASCII REPEATED DIVISIONS BY 10 AND ADDING
THE REMAINDER TO 30 HEX WILL ACCOMPLISH THE CONVERSION */
0213 8B060000      MOV    AX,TEMP
0217 99          CWD
0218 F7F9          IDIV   CX
021A 89060000      MOV    TEMP,AX
22 3    END; ; STATEMENT # 22
021E E9C9FF      JMP    @3
021F @4:
23 2    END BINSDECSASC; ; STATEMENT # 23
0221 5D          POP    BP
0222 C20400      RET    4H
BINDECASC      ENDP

/* CO - EXTERNAL PROCEDURE TO OUTPUT A CHARACTER TO THE SYSTEM CONSOLE.
THIS PROCEDURE IS PART OF THE ISBC 957 LIBRARY FOR CONSOLE I/O
PARAMETER:
CHAR - ASCII CHARACTER TO BE OUTPUT ON THE CONSOLE
*/
24 1    CO: PROCEDURE (CHAR) EXTERNAL;
25 2    DECLARE CHAR BYTE;
26 2    END CO;

/* MATRIX DIMENSIONS */
27 1    DECLARE M LITERALLY '6';
28 1    DECLARE N LITERALLY '5';
29 1    DECLARE P LITERALLY '3';

/* THE THREE MATRICES ARE DECLARED AS ARRAYS OF STRUCTURES. X$ROW IS COMPOSED
OF M STRUCTURES EACH OF WHICH IS COMPOSED OF N INTEGER ELEMENTS. THUS
X$ROW MAY BE THOUGHT OF AS A M X N MATRIX. THE MATRIX WILL BE STORED AS
A ROW-ORDER MATRIX WITH THE ELEMENTS OF EACH ROW STORED IN ADJACENT MEMORY
LOCATIONS. Y$ROW IS DECLARED AS A N X P MATRIX AND Z$ROW AS A N X P MATRIX */
30 1    DECLARE X$ROW(M) STRUCTURE (COL(N) INTEGER);
31 1    DECLARE Y$ROW(N) STRUCTURE (COL(P) INTEGER);
32 1    DECLARE Z$ROW(M) STRUCTURE (COL(P) INTEGER);

33 1    DECLARE (I,J,K,MAX) INTEGER;
34 1    DECLARE MAX$ASC$ARRAY(6) BYTE;
35 1    DECLARE TEXT(*) BYTE DATA ('MAX VALUE = ');

/* INITIALIZE X$ROW SUCH THAT THE FIRST ROW IS SET EQUAL TO P, THE SECOND
ROW EQUAL TO 1, THE THIRD ROW EQUAL TO 2, ETC. */
36 1    DO I = 0 TO (M-1); ; STATEMENT # 36
0002 FA          CLI
0003 2E0E160000      MOV    SS,CS:@STACK$FRAME
0008 BC0000      MOV    SP,@STACK$OFFSET
000B 8BEC      MOV    BP,SP
000D 16          PUSH   SS
000E 1F          POP    DS
000F FB          STI
0010 C70602000000    MOV    I,PH
0016 813E02000500    CMP    I,5H
001C 7E03          JLE    $+5H
001E E93C00      JMP    @7
37 2    DO J = 0 TO (N-1); ; STATEMENT # 37
0021 C70604000000    MOV    J,0H
0027 813E04000400    CMP    J,4H
002D 7E03          JLE    $+5H
002F E92200      JMP    @9
38 3    X$ROW(I).COL(J) = I; ; STATEMENT # 38
0032 8B060200      MOV    AX,I
0036 B92A00      MOV    CX,FAH
0039 F7E9          IMUL   CX
003B 8B360400      MOV    SI,J
003F D1E6          SHL    SI,1
0041 89C3          MOV    BX,AX
0043 8B0E0200      MOV    CX,I
0047 890E0400      MOV    [BX].X$ROW[SI],CX
39 3    END;

```

```

; STATEMENT # 39
004B 010684000100 ADD J,1H
0051 E9D3FF JMP P8
40 2 END;

; STATEMENT # 40
0054 010682000100 ADD I,1H
005A E9B9FF JMP P6
07:

/* INITIALIZE YSROW SUCH THAT THE FIRST COLUMN IS SET EQUAL TO 0, THE
SECOND COLUMN EQUAL TO -1, AND THE THIRD COLUMN EQUAL TO -2. */
41 1 DO I = 0 TO (N-1);

; STATEMENT # 41
005D C70682000000 MOV I,0H
010:
0063 013E82000400 CMP I,4H
0069 7E03 JLE S+5H
006B E94000 JMP P11
42 2 DO J = 0 TO (P-1);

; STATEMENT # 42
006E C70684000000 MOV J,0H
012:
0074 013E84000200 CMP J,2H
007A 7E03 JLE S+5H
007C E92600 JMP P13
43 3 YSROW(I).COL(J) = -J;

; STATEMENT # 43
007F 0B068400 MOV AX,J
0083 F7D8 NEG AX
0085 50 PUSH AX
0086 8B068200 MOV AX,I
008A B90600 MOV CX,6H
008D F7E9 IMUL CX
008F 8B368400 MOV SI,J
0093 D1E6 SHL SI,1
0095 89C3 MOV BX,AX
0097 59 POP CX
0098 89884000 MOV [BX].YROW(SI),CX
44 3 END;

; STATEMENT # 44
009C 010684000100 ADD J,1H
00A2 E9CFFF JMP P12
013:

45 2 END;

; STATEMENT # 45
00A5 010682000100 ADD I,1H
00AB E9B5FF JMP P10
011:

/* PERFORM MATRIX MULTIPLICATION */
46 1 DO K = 0 TO (P-1);

; STATEMENT # 46
00AE C70686000000 MOV K,0H
014:
00B4 013E86000200 CMP K,2H
00BA 7E03 JLE S+5H
00BC E98C00 JMP P05
47 2 DO I = 0 TO (M-1);

; STATEMENT # 47
00BF C70682000000 MOV I,0H
016:
00C5 013E82000500 CMP I,5H
00CB 7E03 JLE S+5H
00CD E97200 JMP P07
48 3 ZSROW(I).COL(K) = 0; /* SET ZSROW ELEMENT TO 0 */

; STATEMENT # 48
00D0 0B068200 MOV AX,I
00D4 B90600 MOV CX,6H
00D7 F7E9 IMUL CX
00DA 8B368600 MOV SI,K
00DD D1E6 SHL SI,1
00DF 89C3 MOV BX,AX
00E1 C7805E000000 MOV [BX].ZROW(SI),0H
49 3 DO J = 0 TO (N-1); /* SUM THE PRODUCT OF XSROW ROW TERMS AND YSROW COLUMN TERMS */

; STATEMENT # 49
00E7 C70684000000 MOV J,0H
018:
00ED 013E84000400 CMP J,4H
00F3 7E03 JLE S+5H
00F5 E94100 JMP P09
50 4 ZSROW(I).COL(K) = ZSROW(I).COL(K) + ( XSROW(I).COL(J) * YSROW(J).COL(K) );

; STATEMENT # 50
00F8 0B068200 MOV AX,I
00FC B90600 MOV CX,6H
00FF F7E9 IMUL CX
0101 8B368400 MOV SI,J
0105 D1E6 SHL SI,1
0107 50 PUSH AX
0108 8B068400 MOV AX,J
010C B90600 MOV CX,6H
010F F7E9 IMUL CX
0111 8B368600 MOV DI,K
0115 D1E7 SHL DI,1
0117 89C3 MOV BX,AX
0119 8B814000 MOV AX,[BX].YROW(DI)
011D 5B POP BX
011E F7A80400 IMUL [BX].XROW(SI)
0122 50 PUSH AX
0123 0B068200 MOV AX,I
0127 F7E9 IMUL CX
0129 89C3 MOV BX,AX

```



```

012B 58 POP AX ; 1
012C 01815E00 ADD [BX].ZROW[DI],AX
51 4 END; ; STATEMENT # 51
0130 810684000100 ADD J,1H
0134 E9B4FF JMP 018 ; STATEMENT # 52
019:
END;
0139 810682000100 ADD I,1H
013F E983FF JMP 016 ; STATEMENT # 53
017:
END;
0142 810686000100 ADD K,1H
0148 E959FF JMP 014 ; STATEMENT # 54
015:
54 1 MAX = FINDSMX (0ZSROW, M, P); /* FIND MAX VALUE OF ZSROW */
; STATEMENT # 54
014B B85E00 MOV AX,OFFSET(ZROW)
014E 50 PUSH AX ; 1
014F B8060F MOV AX,6H ; 2
0152 50 PUSH AX ; 3
0153 B8030F MOV AX,3H
0156 50 PUSH AX
0157 E8000F CALL FINDMX
015A 89068800 MOV MAX,AX
55 1 CALL BINSDECSASC (MAX, #MAXSASCARRAY); /* CONVERT TO DECIMAL ASCII */
; STATEMENT # 55
015E FF368800 PUSH MAX ; 1
0162 B80A0F MOV AX,OFFSET(MAXASCARRAY)
0165 50 PUSH AX ; 2
0166 E84C0F CALL BINDECASC
56 1 DO I = 0 TO (SIGNED(SIZE(TEXT)) - 1); /* OUTPUT HEADER TEXT */
; STATEMENT # 56
0169 C70682000000 MOV I,0H
020:
016F 013E22000B00 CMP I,0BH
0175 7E03 JLE $+5H
0177 E91400 JMP 021
57 2 CALL CO(TEXT(I)); ; STATEMENT # 57
017A 8B1E8200 MOV BX,I
017E FFB70000 PUSH TEXT[BX]; 1
0182 E80000 CALL CO
58 2 ; STATEMENT # 58
0185 810682000100 ADD I,1H
018B E9E1FF JMP 020
021:
59 1 DO I = 0 TO 5; /* OUTPUT ASCII MAX VALUE */
; STATEMENT # 59
018E C70682000000 MOV I,0H
022:
0194 013E22000B00 CMP I,5H
019A 7E03 JLE $+5H
019C E91400 JMP 023
60 2 CALL CO(MAXSASCARRAY(I)); ; STATEMENT # 60
019F 8B1E8200 MOV BX,I
01A3 FFB70000 PUSH MAXASCARRAY[BX]; 1
01A7 E80000 CALL CO
61 2 ; STATEMENT # 61
01AA 810682000100 ADD I,1H
01B0 E9E1FF JMP 022
023:
62 1 END EXECUTIONSVEHICLE; ; STATEMENT # 62
01B3 FB STI
01B4 F4 HLT

```

MODULE INFORMATION:

```

CODE AREA SIZE = 0225H 549D
CONSTANT AREA SIZE = 000CH 12D
VARIABLE AREA SIZE = 0090H 144D
MAXIMUM STACK SIZE = 0008H 8D
137 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

OVERVIEW

This overview is provided to investigate both the problems encountered in the design of applications software and also the classic solutions to these problems.

Multitasking

A real-time system is defined to be a system that reacts to events occurring external to the computer and which monitors or controls these events as they occur (or in "real-time"). The concept of a real-time system is known as a batch system where the outcome of a program does not depend on when it is run (for example a payroll program).

Two other characteristics of a real-time system are: real-time systems are characterized by a high degree of concurrency. The first characteristic is that events occur randomly rather than at scheduled intervals. The second characteristic is that current activity takes place when two or more events occur nearly at the same time, requiring simultaneous activity.

One method of dealing with the requirements of a real-time system would be to write a program that knows what events should potentially occur (for example, an interrupt occurrence, a real-time clock counting down to zero, a byte in memory being modified by another program). This program could then execute a large loop checking for the occurrence of these events.

There are several problems with this approach. While processing one event which has occurred, the program is not responsive to other events. Also, the programmer has no way of prioritizing the importance of the various events. A main response endpoint of this program is complex and difficult to enhance or modify.

The traditional solution to these problems is a technique called multitasking. Essentially, this involves dividing many small routines (tasks) into process events in the system. In addition, each task can be assigned each task so that the operating system can decide as to which task is the most important when more than one task request occurs.

Supporting multitasking involves a scheduler which is provided by the operating system. Each task is executed in a program of the CPU, ensuring that all tasks are serviced according to the priority of each task.

INTRODUCTION

Companies seeking to develop microcomputer applications are faced with two significant problems. First, applications are growing more and more sophisticated. With competition always present, products are continually being enhanced with new features. This puts more and more pressure on the computer system by increasing the complexity of the software and the number of events and functions that must be handled by the system.

The second problem is a management problem. These newer and more sophisticated application systems have developed quickly in order to hit shrinking markets. As a result, there must be a development staff that is not only technically competent but also capable of managing the development costs.

There are the needs addressed by the iRMX 86™ Operating System. The two goals in the development of this product have been power/efficiency to meet the needs of increasingly complex application systems, and ease of understanding and use, to boost the productivity of available engineering resources. Users of Intel's line of iRMX 86™ Single Board Computers or custom-designed iRMX-based boards can now obtain the same benefits from Intel supplied system software as they can from Intel supplied system hardware.

The reader of this application note is provided with information in four subject areas:

- The requirements of operating systems are discussed along with traditional solutions.
- The iRMX 86 Operating System is presented and its features are discussed in relation to the requirements outlined above.
- System design using iRMX 86™ is discussed.
- System is shown using iRMX 86™.
- Code for two example systems is provided to learn the details of system implementation.

Throughout this application note various terms and concepts are introduced. If further information on any of these is desired, the references listed in the front of the note should be consulted.

Steve Versteeg
OEM Microcomputer Systems Applications

July 1980

INTRODUCTION

Companies seeking to develop microcomputer applications are faced with two significant problems. First, applications are growing more and more sophisticated. With competition always present, products are continually being enhanced with new features. This burdens the underlying computer system by increasing both the complexity of the software and the number of events and functions that must be handled by the system.

The second problem is a management problem. These newer and more sophisticated application systems must be developed quickly in order to hit shrinking market windows. Also, they must be developed with lower manpower costs to be feasible in an engineering community struck by insufficient technical personnel and skyrocketing software development costs.

These are the needs addressed by the iRMX 86™ Operating System. The two goals in the development of this product have been power/flexibility to meet the needs of increasingly complex application systems, and ease of understanding and use, to boost the productivity of available engineering resources. Users of Intel's line of iSBC 86™ Single Board Computers or custom-designed 8086-based boards can now obtain the same benefits from Intel supplied system software as they can from Intel supplied system hardware.

The reader of this application note is provided with information in four subject areas.

- The requirements of operating systems are discussed along with traditional solutions.
- The iRMX 86 Operating System is introduced and its features are discussed in relation to the requirements studied earlier.
- System design using the iRMX 86 Operating System is studied using example solutions.
- Code for two example systems is examined to learn the details of system implementation.

Some of the topics in this note may not be of interest to all readers. For example, an experienced real-time programmer may not need to read the entire overview of real-time systems. For those who want to brush up on a few topics, the overview is organized to allow the reader to focus attention on areas of specific interest.

Throughout this application note, various terms and concepts are introduced and discussed. If further information on any of these topics is desired, the references listed in the front of this note should be used.

OVERVIEW

This overview is provided to investigate both the problems encountered in the design of applications software and also the classical solutions to these problems.

Multitasking

A real-time system is defined to be a system that reacts to events occurring external to the computer and which monitors or controls these events as they occur (or in "real-time"). The converse of a real-time system is known as a batch system where the outcome of a program does not depend on when it is run (for example, a payroll program).

Two other characteristics typically encountered in a real-time system are asynchronous event occurrences and concurrent activity. The first characteristic is caused by events occurring randomly rather than at scheduled intervals. The second characteristic, concurrent activity, takes place when two or more events occur nearly at the same time, requiring simultaneous activity.

One method of dealing with the requirements of a real-time system would be to write a program that knows what events could potentially occur (for example, an interrupt occurrence, a real-time clock counting down to zero, a byte in memory being modified by another program). This program could then execute a large loop checking for the occurrence of these events.

There are several problems with this approach. While processing one event which has occurred, the program is not responsive to other events. Also, the programmer has no way of prioritizing the importance of the various events. From a maintenance standpoint, this program is complex and difficult to enhance or modify.

The traditional solution to these problems is a technique called multitasking. Essentially, this involves writing many small routines instead of one large one. Each of these routines (tasks) can process events independent of the other tasks in the system. In addition, a priority can be assigned each task so that the operating system can decide as to which task is the most important when more than one task requests control of the CPU.

The support for multitasking involves a scheduler which is part of the service provided by the operating system. The scheduler allows each task to execute its program as if it has sole control of the CPU, ensuring that all tasks desiring CPU time are serviced according to the priority associated with each task.

From the standpoint of system design, multitasking has many desirable qualities. Large and potentially complex application programs can be decomposed into smaller more manageable units. This makes feasible the use of programmer teams to implement the application. Perhaps even more importantly, the potentially overwhelming problems surrounding concurrent execution and interrupt handling become transparent to the application programmer. Also, multitasking makes the modification of existing tasks and the addition of new ones become a manageable objective since the interaction between tasks is minimized.

Interrupt Handling

A common event in a real-time system is the occurrence of an interrupt. Because this event is so common, an important feature of a real-time operating system is its interrupt processing capabilities.

From the standpoint of application software, interrupt handling can be cumbersome. The currently running task must be preempted, various hardware devices must be manipulated and perhaps a hardware interrupt controller must be dealt with.

A real-time operating system can abstract the occurrence of an interrupt into something more consistent with the way other events are handled. A task can simply inform the scheduler that it does not require any CPU time until an interrupt occurs. The relative priority of different interrupts can also be handled in the same manner as the priority of multiple tasks are handled. Thus, the application programmer need only deal with the actual processing related to interrupt occurrence.

Reliability

Reliability is a keyword in all real-time systems. In this type of system, reliability does not refer to mean time between failure. In fact, the software in a real-time application typically cannot be *allowed* to fail. The difficulty imposed on the software by the environment comes from the near infinite number of permutations that can occur. A system that appears to be fully debugged can fail in the field because of a combination of simultaneous events that never occurred before.

The only means to avoid failure in these instances is through the use of a consistent, well-thought-out model for handling events. Any special-cased solution is subject to failure when the special cases that were designed for are violated in the real world.

Error handling can also add reliability to an application system. When the application software is

unable to anticipate the outcome of certain conditions, or the software has undiscovered bugs, it is vital for the operating system to gracefully handle the situation and allow for further processing to continue as best as possible.

I/O Handling

Many applications for 16-bit microcomputers require a variety of I/O devices. The support for I/O operations on these devices is typically provided by the operating system. Both sequential access and random access devices are typically encountered and, in addition, flexibility in handling I/O requests and acknowledgements is important.

The flexibility necessary typically involves the scheduling of a task's execution after an I/O request has been made. The greatest flexibility can be obtained by an *asynchronous* I/O system. In this system, a task makes an I/O request by calling the operating system. Once the processing of the request has begun, control is returned to the calling task.

In this manner, the task can continue executing its program while the I/O operation is progressing. When the results of the operation are desired, the task can call the operating system again to wait for the completion of the previous I/O request.

The second type of I/O support is less flexible but also easier to use. An operating system that supports *synchronous* I/O allows a task to make a single operating system call to make an I/O request. Once control is returned to the calling task, the I/O operation is complete and the results are immediately available. This type of I/O support sometimes takes advantage of a technique known as *autobuffering* to regain some of the performance advantage of the overlapped I/O found in the asynchronous system.

Debug Support

The inherent characteristics of the real-time environment sometimes make it difficult to debug new software. If the simultaneous occurrence of two events causes a bug in the software, detection may be difficult because the next time the system is run the error is not reproduced. Also, because of the fact that the software is broken down into many independent tasks, the interaction may be difficult to track using standard debugging techniques.

The solution to these problems is a piece of software called the system debugger. The debugger typically has three characteristics.

- 1) It is designed to interact with the operating system and therefore has intimate knowledge of code, data structures and system objects.
- 2) Since the debugger is just another task in the system, it does not affect the operation of the other tasks that are running.
- 3) Through the use of sophisticated breakpointing facilities, the debugger allows the designer to track the tasks in the system, investigate their interaction with other tasks and selectively stop one or more tasks without stopping the entire system.

Multiprogramming

In some application systems, there arises the requirement to run several "applications" on the computer at the same time. This may be due to the desire to squeeze more use out of the hardware or it may be due to some system design consideration. These separate "applications" (often termed jobs) share many system resources (especially the CPU) but at the same time they need to be protected as much as possible from other jobs. In essence, it should be possible to develop two jobs independently and then run them both on the same hardware without any interaction. If interaction is desired, the operating system should support some well-defined protocol for jobs to use to communicate.

Free Space Management

One of the most important resources in the computer system is the memory. In some applications, the amount of memory needed can be determined when the system is designed. In the more general case, the amount of memory needed by the system fluctuates. One solution to this management problem is to have available the amount needed in the worst possible case. A more flexible and economical solution is to dynamically allocate memory from a central pool upon demand and return it when possible. This service provides two tangible advantages. First, total memory needs are reduced. Second, this service allows for ease of use by the application programmer because there is no need to set aside blocks of memory and implement code to maintain information about current usage.

File Management

The ability to easily store and retrieve data stored on mass storage devices is a requirement in many application systems. Devices such as disks, tapes and bubble memories are used to store program code, data files and parameter tables. The operating system is called upon to store and retrieve the data and organize it such that application programs can easily find and manipulate the data when necessary.

Typically, this service is provided through the use of a file system. The mass storage device is partitioned into blocks and logical addresses are assigned to the blocks. Files are created to serve as directories where the names of other files can be cataloged and looked up.

In many systems, the directory structure can go many levels deep (see Figure 1). This provides several advantages. Directory searches can be done much faster if the general area where a file exists is known. Also, if several jobs are running at the same time, each can be given its own directory and therefore isolated from the others. Lastly, for human users, it is much easier to manage the information on the disk when some logical structure of files exists.

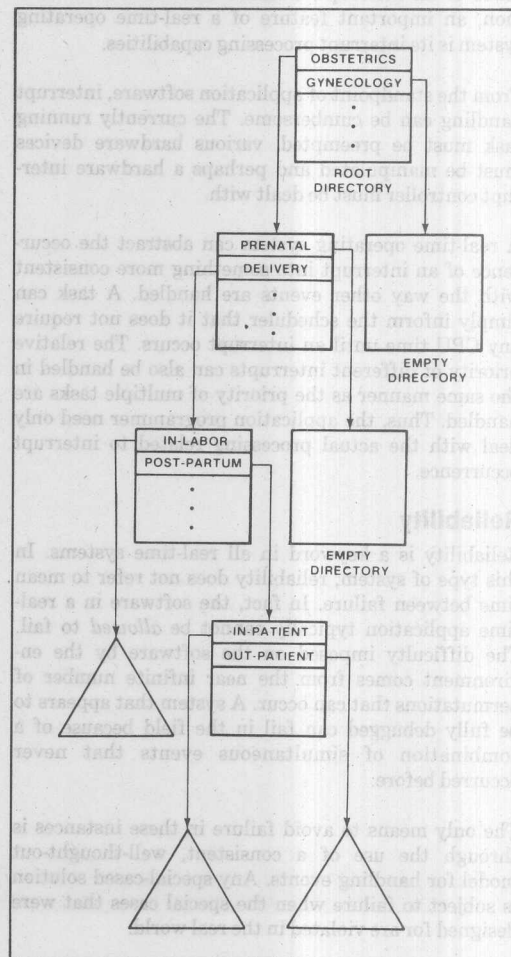


Figure 1. Hierarchical File System

Device Independence

One of the unfortunate characteristics of I/O devices is that they all tend to present different interfaces to the system software. When this is the case, the application programmer must become familiar with the unique characteristics of each device in order to communicate with it. One solution is to create an I/O driver which does the actual I/O. This driver can then be called by the application program whenever communication with the device is desired.

The problem with this solution is that the programmer must still know what type of device is being talked to since the I/O driver is specialized. If the system configuration changes, all of the software must be rewritten to call new device drivers. The best solution is to design a standard interface to device drivers and postpone until run-time the decision about which devices to use. With this type of system, an application program can be written assuming that at run-time the human or program that invokes it will provide a specification of which devices should be used.

High-Level Man-Machine Interface

In addition to the services provided for application programs by the operating system, a set of services typically is offered to the human user sitting at the system console. System utilities are needed for file copying, disk formatting, and directory maintenance. Programs need to be loaded off disk to run and the programs themselves must be able to retrieve parameters passed to them by the operator. All of these functions are usually provided by the man-machine interface software in the operating system.

Make Versus Buy

The previous sections dealt with operating system requirements. These requirements are encountered in the application development process. Whether the solution to meet the needs comes from the individual application designer or from a computer system vendor, the requirements do not change.

There usually exists a rather simple tradeoff between designing a custom operating system or buying a generalized system and tailoring it to the individual needs of the application. There are advantages to the custom solution. The system can often be made smaller since the requirements are known in great detail. Also, some small performance improvements can sometimes be made by taking advantage of the special cases to speed things up.

Buying an operating system from a computer system vendor offers five advantages.

- 1) Engineering resources are becoming scarce. The use of an operating system from a vendor allows attention to be focused on the application software.
- 2) The time taken to bring the product to market can be shortened, thereby gaining a competitive edge and generating early revenue.
- 3) Long-term maintenance costs can be reduced because the vendor supports the operating system software.
- 4) Personnel in all branches of the company can become familiar with one software architecture and apply this knowledge to a range of products. This applies not only to the design engineers, but also to quality assurance, customer engineers and system analysts.
- 5) The computer system vendor has knowledge of future technological advances coming in the product lines. For this reason, the operating system can be constructed so that applications software can be transported to future hardware without the need for expensive redesign.

In summary, the trade-offs are clear. An operating system from a computer system vendor is not the answer for every application. But in most cases, the most economical and safest bet is to take advantage of the expertise of the vendor for the system software and use engineering resources to more quickly solve the application problem.

INTRODUCTION TO THE iRMX 86™ OPERATING SYSTEM

The iRMX 86 Operating System meets the needs of real-time applications while simultaneously providing the full set of services normally found in a general-purpose operating system.

The overall picture of the iRMX 86 Operating System is shown in Figure 2. The iRMX 86 Nucleus provides

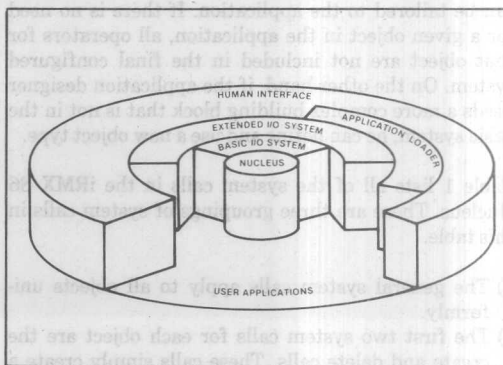


Figure 2. Layers of Support in the iRMX 86™ System

support for multitasking, multiprogramming, inter-task communication, interrupt handling and error checking. The Basic I/O System provides support for device independent and file format independent manipulation of data on I/O devices. The Extended I/O system provides synchronous I/O calls, automatic buffering, logical file name support and high-level job management. The application loader provides the ability to load code and data from mass storage devices into RAM memory. The Human Interface provides for a high-level man-machine interface as well as file utilities and parsing support for application programs.

The following sections deal in more detail with each of these iRMX 86 pieces. If more information is desired on the features discussed, please refer to the documents listed in the front of this application note.

Architecture

The iRMX 86 architecture is an object-oriented architecture. This means that the operating system is organized as a collection of building blocks that are manipulated by operators. The building blocks of the iRMX 86 system are called objects and are of several types. Some of the object types are tasks, jobs, mailboxes, semaphores and segments. These types are explained in subsequent sections of this application note.

This type of architecture has two major advantages. First, the system is easier to learn and use. The attributes of the various objects and the operations that can be performed on them are well defined and consistent. Once an object type is understood, all objects of that type are understood.

The second advantage to an object-oriented architecture is the ease with which the operating system can be tailored to the application. If there is no need for a given object in the application, all operators for that object are not included in the final configured system. On the other hand, if the application designer needs a more complex building block that is not in the basic system, he can define and use a new object type.

Table 1 lists all of the system calls in the iRMX 86 Nucleus. There are three groupings of system calls in this table.

- 1) The general system calls apply to all objects uniformly.
- 2) The first two system calls for each object are the create and delete calls. These calls simply create a new object and initialize its attributes or delete an existing object.

- 3) The remaining system calls are specific to the attributes of a particular object. With this organization in mind, the entire operation of the iRMX 86 nucleus can be glimpsed in a single table.

Tasks

Tasks are the active objects in the iRMX 86 architecture. Tasks execute program code and therefore are the only objects that can manipulate other objects. The attributes of a task include its program counter, stack, priority and dispatcher state.

Tasks compete with each other for CPU time and the iRMX 86 scheduler determines which task to run based upon priorities. The dispatcher states for an iRMX 86 task are shown in Figure 3. At any given point in time, the highest priority task that is ready to run has control of the CPU. Control is transferred to another task only when

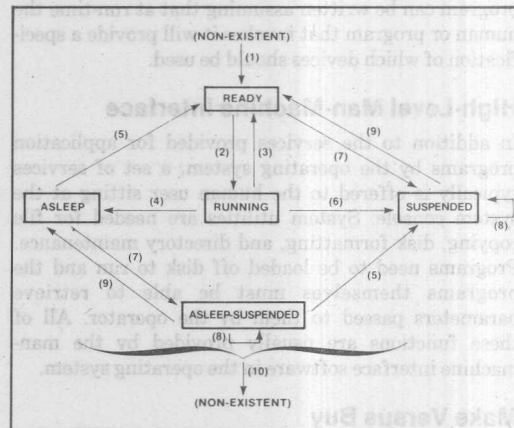


Figure 3. Task State Transition Diagram

- 1) the running task makes a request that cannot immediately be filled and is, therefore, moved to the asleep state,
- 2) an interrupt occurs causing a higher-priority task to become ready to run or
- 3) the running task causes a higher-priority asleep task to become ready by releasing some resource.

The suspended and asleep-suspended states are entered whenever the suspend system call is invoked for a particular task.

Job and Free Space Management

Support for multiprogramming is provided by the job object. A job provides the environment for tasks to execute their programs. All other objects needed for a particular application are contained within the job.

Table 1. Nucleus Object Management System Calls

System Calls for All Objects	O.S. Objects	Attributes	Object-Specific System Calls
	JOB	Tasks Memory pool Object directory Exception handler	CREATE\$JOB DELETE\$JOB SET\$POOL\$MIN GET\$POOL\$ATTRIB OFFSPRING
	TASKS	Priority Stack Code State Exception handler	CREATE\$TASK DELETE\$TASK SUSPEND\$TASK RESUME\$TASK GET\$EXCEPTION\$HANDLER SET\$EXCEPTION\$HANDLER SLEEP GET\$TASK\$TOKENS GET\$PRIORITY SET\$PRIORITY
CATALOG\$OBJECT UNCATALOG\$OBJECT			
LOOKUP\$OBJECT	SEGMENTS	Buffer with length	CREATE\$SEGMENT DELETE\$SEGMENT GET\$SIZE
ENABLE\$DELETION DISABLE\$DELETION FORCE\$DELETE	MAILBOXES	List of objects List of tasks waiting for objects	CREATE\$MAILBOX DELETE\$MAILBOX SEND\$MESSAGE RECEIVE\$MESSAGE
GET\$TYPE	SEMAPHORES	Semaphore unit value List of tasks waiting for units	CREATE\$SEMAPHORE DELETE\$SEMAPHORE RECEIVE\$UNITS SEND\$UNITS
	REGIONS	List of tasks waiting for critical section	CREATE\$REGION DELETE\$REGION RECEIVE\$CONTROL ACCEPT\$CONTROL SEND\$CONTROL
	USER OBJECTS	License rights to a given extension type New object template	CREATE\$EXTENSION DELETE\$EXTENSION CREATE\$COMPOSITE DELETE\$COMPOSITE INSPECT\$COMPOSITE ALTER\$COMPOSITE

A specific attribute of the job is a free memory pool from which blocks can be allocated only by tasks within the job. Also, the job contains an object directory which can be used by tasks to catalog objects under ASCII names so that other tasks, knowing the ASCII name, can look up the object and thereby gain addressability to it.

More than one job can co-exist in the computer system. Tasks within jobs can also create children jobs forming a hierarchical tree of jobs (see Figure 4). Each job in the system has its unique set of contained objects, its own memory pool and its own object directory.

Segments

A fundamental resource that tasks need is memory. Memory is allocated to tasks in the form of the

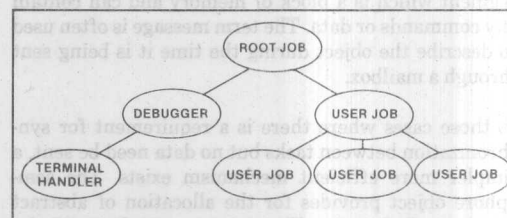


Figure 4. iRMX 86™ Job Tree Example

segment object. The segment is a block of contiguous memory. The attributes of a segment are its base address and size. A task needing memory requests a segment of whatever size it requires. The Nucleus attempts to create a segment from the memory pool given to the task's job when the job was created.

If there is not enough memory available, the Nucleus will try to get the needed memory from ancestors of the job.

Communication and Synchronization

In many cases it is necessary for two tasks to communicate in order to exchange data and commands. This is supported through the use of an object known as a mailbox. As its name implies, a mailbox is a holding place for objects. One task can send an object to a mailbox, causing the object to be queued there. Another task can later receive an object from the mailbox and thereby gain access to it (see Figure 5). If a task tries to receive an object from a mailbox and there are no objects there, the task can optionally be made to sleep for a specified time for an object to appear.

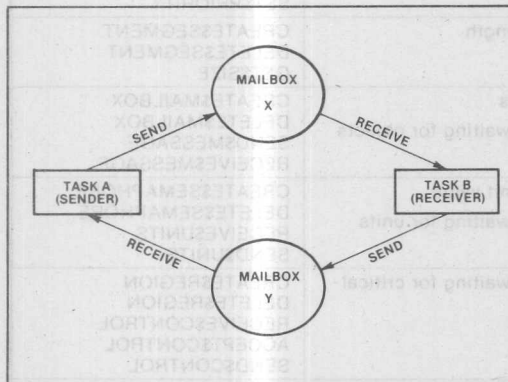


Figure 5. Intertask Communication via Mailboxes

Note that any object can be sent to a mailbox to be received by another task. Typically, the object sent is a segment which is a block of memory and can contain any commands or data. The term message is often used to describe the object during the time it is being sent through a mailbox.

In those cases where there is a requirement for synchronization between tasks but no data need be sent, a simpler more efficient mechanism exists. The semaphore object provides for the allocation of abstract entities called units. The primary attribute of the semaphore is an integer number. Tasks may send units to a semaphore thereby increasing the integer number or they can request units, thereby decreasing the number. If a task makes a request for more units than are available, it can optionally be made to sleep for a specified amount of time. This mechanism can be used for synchronization, resource allocation and mutual exclusion.

Interrupt Management

When an interrupt is sensed by the 8086 hardware, a user interrupt handler is executed. The interrupt handler can either perform all interrupt processing itself without making any iRMX 86 system calls, or it can signal an interrupt task allowing more general interrupt processing including calls to the operating system.

The operating system maps hardware interrupt priorities into the software priority scheme allowing the designer to specify what software functions are important enough to have some interrupt levels masked off during their execution. Although this mapping should always be kept in mind during design, the mechanics of dealing with interrupt control are handled by the operating system.

Error Management

One of the central themes in the design of the iRMX 86 operating system has been reliability. The results of these efforts are evident in two particular features of the architecture. Beyond the ease of understanding brought about by the symmetry of the system, the reliability of applications using the iRMX 86 software is increased.

The general case (as opposed to checking only for specific combinations of errors) has been designed for. Because of this, an unexpected combination of events or the simultaneous occurrence of interrupts will never catch the system by surprise.

In the event that errors do occur, the operating system is set to detect them. Virtually all parameters in calls to the operating system are checked for validity. Any inconsistency causes a jump to an error routine to handle the problem. Two types of errors can potentially occur and there are two ways of handling errors.

The first error type is the programmer error condition which comes about due to some mistake in the coding of a system call. The second type is an environmental condition which arises due to factors out of the control of the engineer (e.g. insufficient memory). Each of these error types can be handled in-line by checking a status code upon return from the call or can cause an error handling subroutine to be called by the system. The system designer can choose the desired method for the system, for a specific job, and even for individual tasks within a job.

Asynchronous I/O

Asynchronous I/O system calls are provided to support device independent I/O to any device in the

system. The type of I/O and the type of device are interrelated as shown in Figure 6. Every device driver in the I/O system is required to support a standard interface. In this manner, all devices look the same to higher level software. In the same manner, the individual file drivers, which provide the different types of file systems, all have a standard interface and call upon the various device drivers to perform I/O. These interface standards

- 1) provide for the device independence in the higher layers of the I/O system
- 2) make it easier for Intel to add future device drivers as new devices become available and
- 3) make it possible for iRMX 86 users to add their own drivers for custom I/O devices.

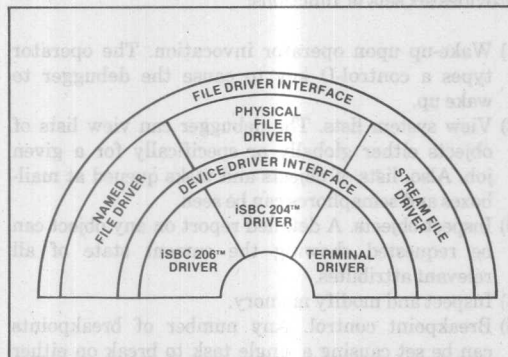


Figure 6. I/O System Structure

The iRMX 86 I/O system provides both asynchronous and synchronous system calls. The asynchronous I/O calls are faster, provide more flexibility in the selection of options and allow the program making the call to perform other functions while waiting for the I/O operation to complete.

The method by which the I/O system responds to the requestor is through the use of a mailbox. When any call is made to the asynchronous I/O system, one of the parameters indicates a mailbox where the caller expects to receive a segment containing the results of the operation (see Figure 7).

Synchronous I/O

The alternative to using the asynchronous I/O system is to use synchronous I/O system calls. As shown in Figure 8, the number of options available are fewer and the caller cannot continue execution until the entire I/O operation is completed but from an ease-of-use standpoint, the situation is much simplified.

```
Response$mailbox$token = RQ$create$
mailbox (0, @status);
CALL RQ$A$read(connection$token, buf$ptr,
count, response$mailbox$token, @status);
IORS$token = RQ$receive$message
(response$mailbox$token, OFFFHH,
@resp$t, @status);
{check status}
Call RQ$delete$segment(IORS$token,
@status);
```

Figure 7. Asynchronous I/O Call

```
Call RQ$$$read(connection$token, buf$ptr,
count, @status);
{check status}
```

Figure 8. Synchronous I/O Call

Two other features provided by the Extended I/O System are logical name support and autobuffering. Logical names allow the application designer to postpone the decision concerning which files to use until run-time. Essentially, all programs can be written and compiled using logical file names and then these logical names can be mapped into real file names at run-time.

The use of autobuffering regains much of performance advantage offered by overlapped I/O. When a user task opens a file for input, one or more buffers are automatically created and filled with data from the file. Thus, when the user task makes an I/O request, the data may already be available in memory. A similar case exists for write requests in that the I/O system will buffer data to be written to a device, allowing the user task to continue on.

Loaders

The iRMX 86 application loader and bootstrap loader perform a variety of services for the user software. The following is a brief summary of the available features.

- 1) Systems can be boot loaded from mass storage devices at system reset. This saves not only ROM or EPROM memory, but also reduces field maintenance costs by allowing easy field updates.
- 2) Users can design their own SYSGEN procedure allowing tailoring of an application system to the individual installation.
- 3) Infrequently used programs can be brought in from mass storage when needed instead of using system memory unnecessarily.

File Management

There are three types of files supported by the iRMX 86 I/O system, named files, physical files and stream files. Named files are supported on devices possessing mass storage capability. Files in this system have ASCII pathnames and are cataloged in directories. Each device in the system contains a directory tree as shown previously in Figure 1. Access protection is provided through the use of access lists for each file. Each user or group of users in the system can be given different types of access to the file or can be denied access to it.

For devices that cannot support a named file structure (e.g. printers and terminals) the physical file driver is used. Devices in this category are treated strictly as data going into and/or out of the device. If it is desirable to treat a mass storage device strictly as a large mass of data, it can also be addressed through the physical file driver.

The third type of file is the stream file. This file type has no correlation with any physical device but rather uses system memory for temporary storage of data. An example of the usage of a stream file is a job that gets its input stream of data from a file. Depending on which time the job is run, this file might be a named file on disk, a terminal, or a stream file being written to by another job (see Figure 9).

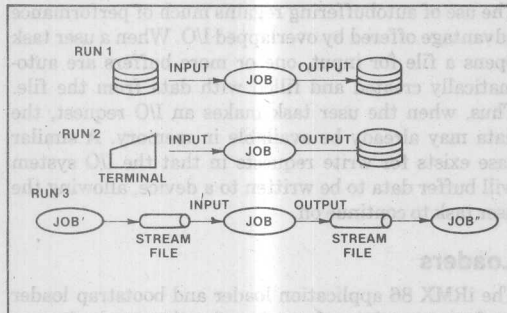


Figure 9. Stream File Example

Human Interface Subsystem

The highest level of support provided by the iRMX 86 Operating System is the Human Interface Subsystem. This piece of software provides two basic services. Programs can be invoked by typing the program name at the system console. The Human Interface will load the given program into memory, set it up as a job and start it running. The invoked program can then call upon the Human Interface routines to determine what parameters were passed to it as part of the operator input.

The Human Interface also contains a set of system utility routines which are used to copy files and disks, format disks, dynamically alter the system configuration and others.

Debugging Subsystem

The iRMX 86 Debugging Subsystem allows the designer to interact with the prototype system and isolate and correct program errors. Since the debugger is an object-oriented debugger and is aware of the internal structure of the operating system, it can provide detailed information concerning objects and can monitor mailboxes and semaphores providing a breakpoint facility as well as error detection.

Specifically, the iRMX 86 Debugging Subsystem provides six sets of functions:

- 1) Wake-up upon operator invocation. The operator types a control-D key to cause the debugger to wake up.
- 2) View system lists. The debugger can view lists of objects either globally or specifically for a given job. Also, lists of objects and tasks queued at mailboxes and semaphores can be seen.
- 3) Inspect objects. A detailed report on any object can be requested showing the current state of all relevant attributes.
- 4) Inspect and modify memory.
- 5) Breakpoint control. Any number of breakpoints can be set causing a single task to break on either execution of particular instructions or sends and receives of messages or units.
- 6) Error handling. The debugger can be set up to be the system default error handler thus catching system exceptions.

Configuration and Initialization

Once the application is designed and coded, the engineer needs a mechanism to inform the operating system of the software and hardware configuration. Essentially, this involves building tables of information using tools provided with the iRMX 86 product.

As shown earlier in Figure 4, the jobs in an iRMX 86 system form a hierarchical tree. The root in every job tree is known as the root job and is supplied as part of the iRMX 86 system. There are three important features of this job.

- 1) The root job has an object directory for cataloging and looking up objects. The special feature of this directory is that it is accessible by all tasks in the system since everyone can address the root job. For this reason the root object directory is useful for setting up inter-job communication paths.

- 2) The root job initially contains all free space in the system. Part of the system initialization code performs a memory scan to automatically determine the amount of free RAM in the system. This memory is put into the free space pool of the root job and parceled out as user jobs are created.
- 3) The root job contains only one task, the root task. This task scans the configuration tables generated by the user and creates the user-specified jobs.

Examples of configuration, initialization and the LINK 86 and LOC 86 operations needed to generate a system will be presented in the Code Examples section.

DESIGN METHODOLOGY

This section describes the design process involved in using the iRMX 86 system to solve application problems and presents two example solutions.

System design with the iRMX 86 Operating System should be viewed as a process starting with the highest level definition of system requirements and successively adding more detail until the end product is program code. This description sounds very much like the description of top-down design and, of course, it should. This methodology offers not only quicker designs, fewer design flaws and easier implementation, but also easier maintenance and enhancement.

In general, every iRMX 86 design progresses through the following steps:

- 1) Define system requirements.
- 2) Breakdown into highest level sub-functions (jobs).
- 3) Define job functions.
- 4) Determine inter-job command and data flow.
- 5) Break down each job into sub-functions.
- 6) Based upon requirements, assign tasks to perform job functions.
- 7) Determine inter-task command and data flow.
- 8) Write program code for each task.

Step 8 becomes the design process associated with the application programs themselves. The code for each task is essentially a sequential program that performs one of the functions of the computer system. Standard techniques for top-down design can therefore be used here to specify each module and its inputs and outputs as well as global and local data structures etc. The end product of this procedure is a modularized application system that should be easy to debug.

APPLICATION EXAMPLE 1

The first example presented here is based on the distributed local network diagrammed in Figure 10. Each

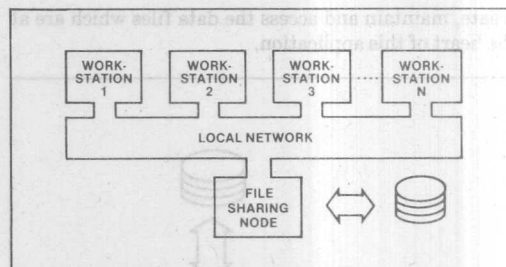


Figure 10. Block Diagram of Example System 1

workstation shown is an intelligent terminal having local data and program storage. The stations all use the File Sharing Node (FSN) for storage and retrieval of records in much the same way as the secretaries in an office would make use of a filing cabinet. The FSN maintains the files on a fixed disk device and responds to requests from the workstations for access to the data. The design to follow concentrates on the File Sharing Node.

System Requirements

Each intelligent terminal in the network has command processing software. When a file reference is made that cannot be satisfied by the local file system, a request is made to the File Sharing Node. This request consists of a log-on request followed by a string of I/O requests and ultimately a log-off request.

The number of intelligent terminals (workstations) hooked up to the FSN varies from installation to installation. Therefore, the FSN must be capable of handling many simultaneous requests and no assumptions can be made about the maximum number of workstations or requests that may need to be handled.

Each node in the network has a unique address. A packet is sent onto the network by one node and the address field is examined by all other nodes. If this field does not match the node's address the packet is ignored. If a match is found the packet is retrieved from the network.

Hardware Requirements

The three main hardware building blocks needed by this application are shown in Figure 11. The iSBC 86/12A Single Board Computer will communicate with the iSBC 544 Intelligent Communications Controller to establish and maintain communications with the network. The Intel 8085A on the iSBC 544 board will perform all of the address recognition, acknowledgements, packet retrieval and packet transmittal. The iSBC 206 Hard Disk Controller will be used to

create, maintain and access the data files which are at the heart of this application.

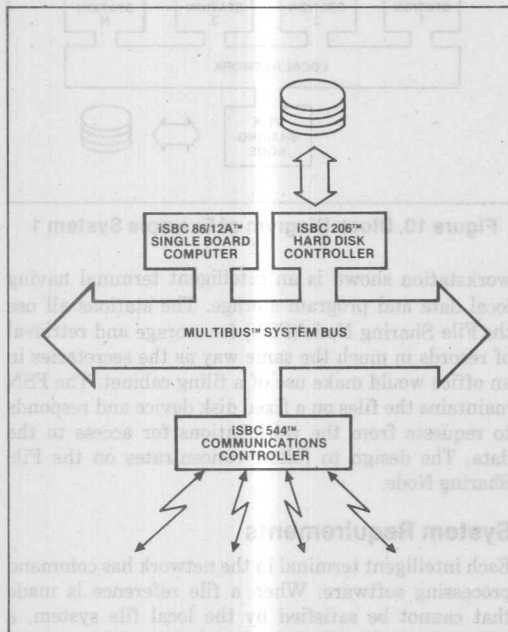


Figure 11. Hardware Block Diagram

System Design

The first step in the system design process is the breakdown of the system functions into one or several jobs. The reasons for doing this are system modularity and protection. With this type of design, each job can be designed separately, perhaps even by a different engineer or engineering team. The input and output requirements will be specified very tightly and the job will take on the appearance of a black box to other jobs in the system. If the job is enhanced or modified at a later date, the rest of the system can be left undisturbed providing that the input and output response remains the same.

The job object in the iRMX 86 operating system also affords a degree of software protection for the tasks and other objects contained within the job. Each job has a separate memory pool, a separate object directory and a separate identification to the I/O system.

The two primary groupings of functions in this application are those related to the network communications and those related to processing the file transaction request. A list of a possible split-up of system functions is shown in Figure 12.

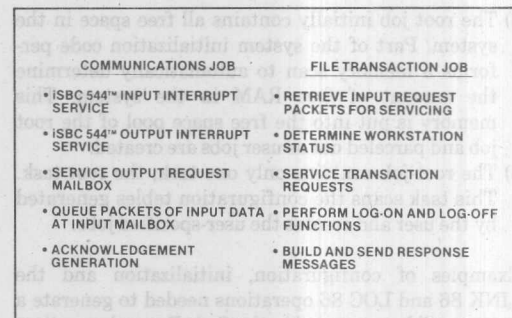


Figure 12. Function Split-up

The communication between the file transaction job and the communication job must fulfill two basic needs. The communication job will receive interrupts when packets addressed to the FSN are received. In order to remain attentive to new requests coming in, the communications job should have the capability to "spool" the requests off to the file transaction job. This buffering can be provided by using the mailbox object. Segments can be created to contain the packet request data and can then be sent to a mailbox where the file transaction job can receive and process them.

When the file transaction job must send a packet to a workstation, the requirement is seen for another queue of requests. Since the communications board can only put one packet at a time on the network, a mailbox should be provided to allow tasks in the file transaction job to send output request segments into the queue and then continue on (see Figure 13).

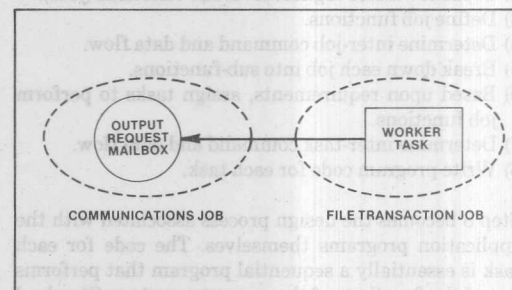


Figure 13. Output Mailbox Queue

Since tasks in both the file transaction job and the communications job must have access to these input and output mailboxes, some means must be set up to "broadcast" the identifier for these objects.

In the iRMX 86 system, each object has associated with it a 16-bit number called a token. Whenever an object is referenced in an operating system call, the

token for the object is used. For example, assume that a segment must be sent to a mailbox. The segment and mailbox each have a token and these tokens are passed to the operating system as parameters in the *send\$message* system call.

There are three major ways to get the token for an object. The first way is to create an object. Whenever the operating system is called to create a new object, the value returned from the procedure call is the token for the new object. The second way to receive a token is through the receive message system call where an object is received from the queue at a mailbox where it was sent by another task.

The third major mechanism for the receipt of a token is provided by the object directory concept. As mentioned previously, each job in the system has an object directory.

If a task in a job has the token for an object and wishes to let other tasks in other jobs have access to the object, the task can "catalog" the object in the object directory. The *catalog\$object* system call takes the token for an object and an ASCII name as parameters and creates an entry in the object directory. If another task knows the ASCII name for an object, it can obtain the token by performing a *lookup\$object* call.

The object directory mechanism will be used in this example to allow the communications job to "broadcast" the tokens for the input and output mailboxes. The jobs for this application are shown in Figure 14.

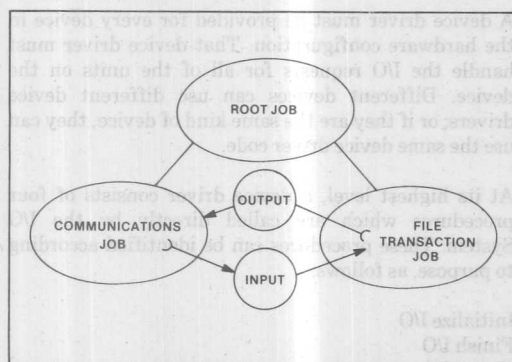


Figure 14. Job Structure

The next step of the design methodology calls for each job to be further divided into sub-functions. In this application note, only the file transaction job is studied.

In time sequence, the file transaction job will:

- 1) Retrieve input requests from the mailbox set up by the communication job.
- 2) Determine state of specified workstation (for example, is it logged on?).
- 3) Perform I/O operation or log-on or log-off.
- 4) Build and send response to the workstation.

Recall from the discussion of system requirements that the number of nearly simultaneous requests that may be received by the FSN is not known. For this reason, some mechanism must be provided to allow parallel processing of many requests. This should prove feasible since the performance of step 3 will involve many delays while waiting for the operating system to perform I/O operations.

One straightforward way to provide for parallel processing is to create a task for each workstation that logs on. In this manner, each I/O request will be handled by a unique task. Through the use of the iRMX 86 scheduler, maximum CPU utilization will be gained by allowing each task to individually compete for CPU time. These "worker" tasks fulfill function 3 and 4 for the file transaction job.

Function 1 and 2 can be fulfilled by a single task. This task will wait at the input mailbox set up by the communications job. When a packet is received that requests a log-on operation, the "listener" task will create a new "worker" task to handle the request. Figure 15 shows a picture of the design.

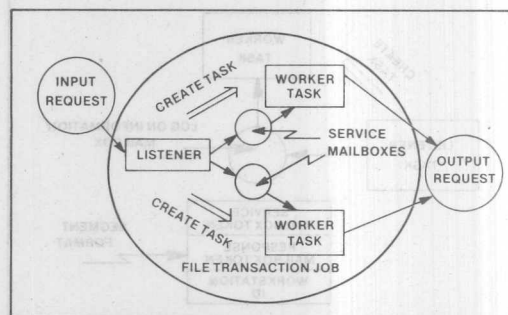


Figure 15. Diagram of Design of File Transaction Job

The string of transaction requests that follow will simply be demultiplexed by the listener task. The workstation ID will be searched for and, if found, the packet will be sent to the appropriate worker task. If a request comes in from a station that is not logged on, an error response is sent directly to the communications output mailbox for transmittal to the station that made the request.

If the request packet indicates that a station desires to log-off, the listener task will delete all local reference to the station and pass the packet along. The listener task cannot simply delete the worker since the worker may be in the process of servicing a previous I/O request. In general, it is never a good idea to arbitrarily delete another task. A better protocol is to pass along the message signaling the worker task to delete itself when convenient.

An investigation of the intertask communications needs highlights the requirement for passing data between tasks. The interjob communications protocol discussed earlier specified that the listener task will receive input request segments from the communications job via a mailbox.

Within these segments are fields containing the workstation ID and the command. Based upon these fields one of two things happens. If the command indicates that the station wishes to log on, a new worker task must be created to process the I/O requests that will follow.

The code executed by all worker tasks will be identical since they all perform identical functions. However, some unique pieces of information must be passed to a new worker task. This can be accomplished by having the worker task first wait at a "log on" mailbox. Here it will receive a segment from the listener task which contains the necessary information (see Figure 16).

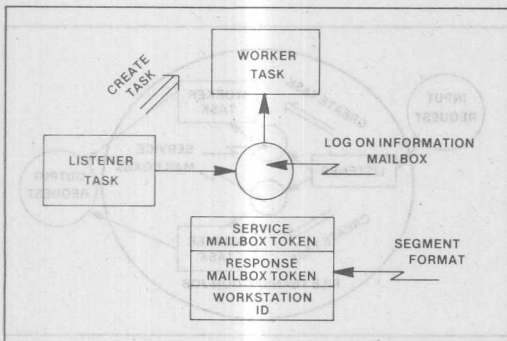


Figure 16. Communications Between Listener Task and a Newly Created Worker Task

After this initialization is complete, the workstation requests that are received by the listener task can be sent to the service mailbox associated with the workstation. The token for the service mailbox is one of the pieces of information contained in the log on segment.

The last communication path needed is predefined by the interjob communication protocol. When either the

listener task or one of the worker tasks needs to transmit a packet to a workstation, a segment is sent to the output request mailbox of the communication job.

The final step in the design methodology is to write program code for the tasks in the system. This step is performed in the Code Examples section.

APPLICATION EXAMPLE 2

This example will deal with the design of a custom device driver for the iRMX 86 operating system. As shown in Figure 6, a device driver accepts high-level commands from the file drivers (such as read, write, seek, etc.) and transforms these commands into I/O port read and write commands in order to communicate with the device itself. By studying the construction of a driver for the iSBC 534 Serial Communication Expansion Board, a better understanding of the iRMX 86 I/O system will be gained along with an example of the use of nucleus facilities to construct a higher-level software function.

Overview of Device Driver Construction

Each I/O device consists of a controller and one or more units. A device as a whole is identified by a device number. Units are identified by unit number and device-unit number. The unit number identifies the unit within the device and the device-unit number identifies the unit among all the units on all of the devices.

A device driver must be provided for every device in the hardware configuration. That device driver must handle the I/O requests for all of the units on the device. Different devices can use different device drivers; or if they are the same kind of device, they can use the same device driver code.

At its highest level, a device driver consists of four procedures which are called directly by the I/O System. These procedures can be identified according to purpose, as follows:

- Initialize I/O
- Finish I/O
- Queue I/O
- Cancel I/O

When a user makes an I/O System call to manipulate a device, the I/O System ultimately calls one or more of these procedures, which operate in conjunction with an interrupt handler to coordinate the actual I/O transfers. This section provides a general description of each of these procedures, and the interrupt handler.

INITIALIZE I/O

This procedure creates all of the iRMX 86 objects needed by the remainder of the routines in the device driver. It typically creates an interrupt task and a segment to store data local to the device. It also performs device initialization, if any such is necessary. The I/O System calls this routine just prior to the first attach of a unit on the device (the first RQ\$A\$PHYSICAL\$ATTACH\$DEVICE system call). The time sequence of calls to these procedures will be described a little later.

FINISH I/O

The I/O System calls this procedure after all units of the device have been detached (the last RQ\$A\$PHYSICAL\$DETACH\$DEVICE system call). The *finish\$I/O* procedure performs any necessary final processing on the device and deletes all of the objects used by the device handler, including the interrupt task and the device-local data segment.

QUEUE I/O

This procedure places I/O requests on a queue, so that they can start when the appropriate unit becomes available. If the device is not busy, the *queue\$I/O* procedure starts the request.

CANCEL I/O

This procedure cancels a previously queued I/O request. Unless the device is such that a request can take an indefinite amount of time to process (such as keyboard input from a terminal), this procedure can perform a null operation.

INTERRUPT HANDLERS AND INTERRUPT TASKS

After a device finishes processing an I/O request, it sends an interrupt to the iRMX 86 system. As a consequence, the interrupt handler for the device is called. This handler either processes the interrupt itself or signals an interrupt task to process the interrupt. Since an interrupt handler is limited in the types of system calls that it can make, an interrupt task usually services the interrupt. The interrupt task feeds the results of the interrupt back to the application software (data from a read operation, status from other types of operations). It then gets the next I/O request from the queue and starts the device processing this request. This cycle continues until the device is detached. The interrupt task is normally created by the initialize I/O procedure.

The I/O System calls each one of the four device driver procedures in response to specific conditions. Three of the procedures are called under the following conditions.

- 1) In order to start I/O processing, the user must make an I/O request. This can be done by making a variety of system calls. However, the first I/O request to each device-unit must be the RQ\$A\$PHYSICAL\$ATTACH\$DEVICE system call.
- 2) The I/O System checks to see if the I/O request results from the first RQ\$A\$PHYSICAL\$ATTACH\$DEVICE system call for the device (the first unit attached in a device). If it is, the I/O System realizes that the device has not been initialized and calls the initialize I/O procedure first, before queueing the request.
- 3) Whether or not the I/O System called the initialize I/O procedure, it calls the queue I/O procedure to queue the request for execution.
- 4) The I/O System checks to see if the request just queued resulted from the last RQ\$A\$PHYSICAL\$DETACH\$DEVICE system call for the device (detaching the last unit of a device). If so, the I/O System calls the finish I/O procedure to do any final processing on the device and clean up objects used by the device driver routines.

The I/O System calls the fourth device driver procedure, the cancel I/O procedure, under the following conditions:

- If the user makes an RQ\$A\$PHYSICAL\$DETACH\$DEVICE system call specifying the hard detach option, in order to forcibly detach the connection objects associated with a device-unit.
- If a job containing the task which made the request is deleted.

Each procedure will now be discussed in more detail. The initialize \$I/O procedure takes three parameters:

init\$I/O: Procedure (*duib\$p*, *ret\$data\$t\$p*, *status\$p*)

The *duib\$p* parameter contains a pointer to a device unit information block (DUIB) which is the configuration table for the device in question. The structure of this table is shown in Figure 17. Note that this table contains pointers to device and unit information tables which can contain hardware specific information (such as I/O base addresses, interrupt levels etc.).

The second parameter is a pointer to a word which can be assigned the value of a token for an iRMX 86 object. Quite often this object would be a segment which could be created by the *init\$I/O* procedure and filled with information needed by the other procedures in the driver. The token for this segment will be provided to the other procedures when they are called.

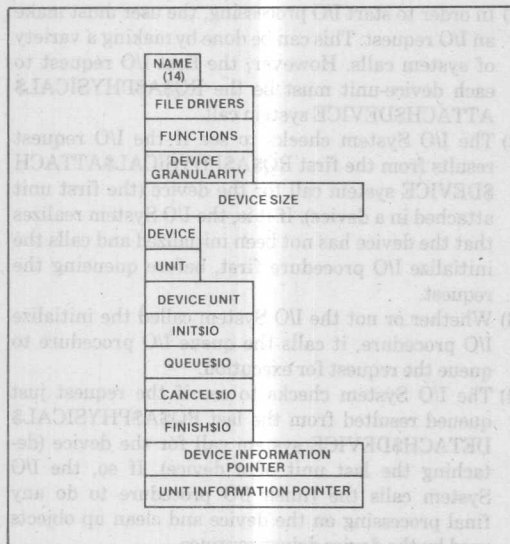


Figure 17. DUIB Format

The final argument in the call is a pointer to a status word. This word should be assigned by the *init\$io* procedure before a RETURN is executed. If a non-zero value is returned indicating an error condition, the I/O System assumes that *init\$io* has deleted any objects created before the error was encountered.

The *finish\$io* procedure is called by the I/O System just after the last *detach\$device* call is made on the device. This procedure is expected to delete any objects created by the *init\$io* procedure and shut down the connected device.

finish\$io: Procedure (duib\$p, ret\$data\$t);

Once again, the first parameter to the call is a pointer to a DUIB. The second parameter is the token returned by the *init\$io* procedure.

The *queue\$io* procedure is called to initiate an I/O request.

queue\$io: Procedure (IORS\$t, duib\$p, ret\$data\$t)

The specifics of the request are indicated in an I/O request segment (IORS) which is provided by the first parameter. The format of this segment is shown in Figure 18. The most important fields here are the count, function, status and buffer pointer fields which tell the *queue\$io* procedure what needs to be done. The second and third parameters are once again the pointer to the DUIB and the token for the object

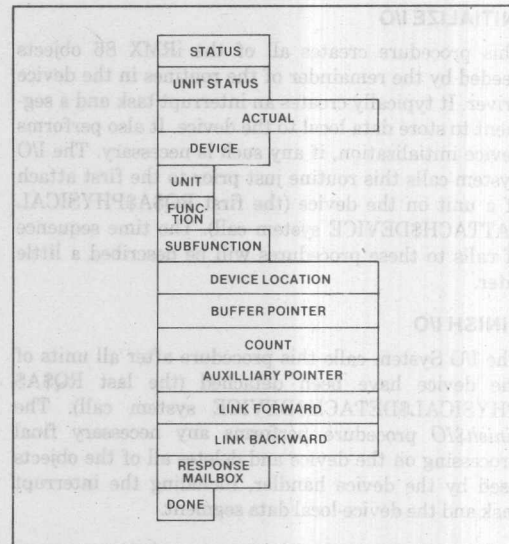


Figure 18. I/O Request Segment Format

created by the *init\$io* procedure.

The final device driver procedure is *cancel\$io*. This procedure is called by the I/O System to cancel a previous I/O request. If the device is of such a nature that a request will complete in a bounded amount of time, this procedure can be a null procedure. The parameters to the call are identical to those for the *queue\$io* call.

In addition to the elementary support discussed here, the I/O System provides extra support to the designer of a device driver if some simplifying assumptions about the device can be made. Also, if the device supports random access (such as disks, magnetic bubbles, etc.), support routines can be used to simplify the process of blocking and deblocking I/O requests. More detail on the process of writing I/O drivers can be found in the manual titled "A Guide to Writing Device Drivers for the iRMX 86 I/O System."

Design of an iSBC 534™ Device Driver

The following section will discuss an example device driver for the iRMX 86 Operating System. The driver will be for the iSBC 534 board which contains four 8251 USART devices; therefore, there is one device and four units on the device.

The *init\$io* procedure for this driver initializes the hardware, creates an interrupt task, creates other necessary objects and creates a segment to contain the relevant information.

The structure of the *queue\$io* procedure is more complex. When calls are made to this procedure to perform data reading and writing, the actual operation could be somewhat lengthy (especially an input operation). Since the *queue\$io* procedure is called by the I/O system, it is not efficient to perform the entire operation before control is returned to the I/O system.

A more efficient mechanism is to have an independent task take the request and fulfill it while the *queue\$io* procedure returns to the I/O system allowing other operations to be started in parallel. This leads to the structure diagrammed in Figure 19. When a read or a write request is received, the I/O request segment is sent to the request mailbox where it is received by an I/O handler task. When the request is complete, the I/O task sends the segment to the response mailbox indicated in the segment.

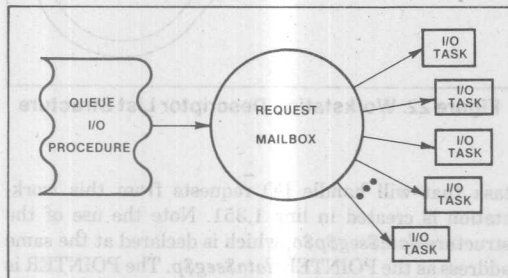


Figure 19. Queue\$io Procedure Interface to I/O Tasks

The remaining design of the device driver is concerned with interrupt handling. The iSBC 534 board contains four 8251 USART devices. Each device supplies two interrupts; one indicating that the receiver has a data character available and the other indicating that the transmitter is ready to accept a character. Each of these interrupts (8 in all) are connected to one of the 8259 Interrupt Controllers on the board. The software on the iSBC 86/12A board must read a register in the 8259 controller to determine which of the eight sources caused the current interrupt. This information must then be fed to the I/O task which may be waiting for the event.

One way to meet this requirement uses an interrupt task for the iSBC 534 board. The task receives the interrupt, determines which device caused it, and sends a unit to a semaphore to indicate the occurrence of the event. Thus, when an I/O task wishes to be informed of a receiver or transmitter interrupt, it simply tries to receive a unit from the appropriate semaphore. If a unit is available, the receiver has a character or the transmitter is ready. If the unit is not

available, the USART is not ready and the task will be put in the asleep state until the interrupt occurs and the unit is sent.

CODE EXAMPLES

This chapter will present and analyze some sample code for the iRMX 86 applications presented in Chapter 4. The code listings are contained in Appendix A and the individual modules are numbered sequentially. When a specific line or sequence of lines of code must be pointed out in the text, a two part number is used where the first part is the module number and the second is the compiler-assigned line number. For example, 3.27 would be used to point out line 27 in module 3.

A standard set of suffixes to labels will be followed in the code to follow. A PL/M-86 WORD variable that will contain the token for an iRMX 86 object will have the suffix "\$t." A POINTER variable will be followed by "\$p" and a structure used to overlay a POINTER allowing access to the base and offset will be followed by "\$p\$o."

Listener Task

The first module to be studied contains the code for the listener task. The various include statements bring in literal declarations and external procedure declarations. The file NUCPRM.EXT is on the iRMX 86 diskette and contains the external declarations for all iRMX 86 nucleus system calls.

Line 1.323 contains all of the declarations for the module. The literal *req\$segment\$struc* is used to access the fields of a segment returned from the communications job. The format of a request packet from a workstation is shown in Figure 20. The literal *node* is used to access the information in a segment used as a workstation descriptor in a list maintained by the listener task. The format of a *node* in this list is shown in Figure 21. The structure at the end of the declaration statement is used to individually access the two halves of a 32-bit PL/M-86 POINTER.

Note in line 1.330 that the task is coded as a public procedure having no parameters. A main procedure should never be used for a task's code since the preamble for a main procedure sets the stack pointer.

The mailbox to be used for sending a newly created worker task an information segment is called the *log\$on\$info\$mbx*. This mailbox is created in line 1.331. Lines 1.332-1.334 perform the operation of finding the tokens for the communication job's input and output request mailboxes in the object directory of

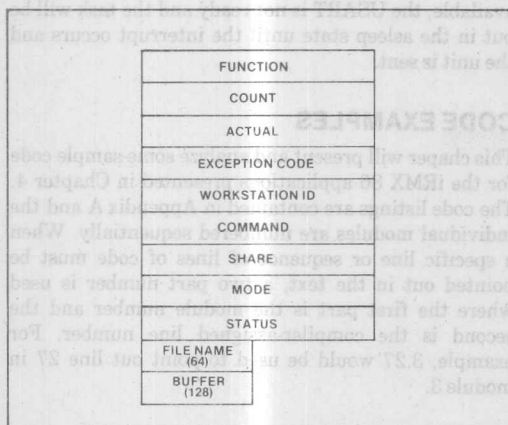


Figure 20. Request Packet Format

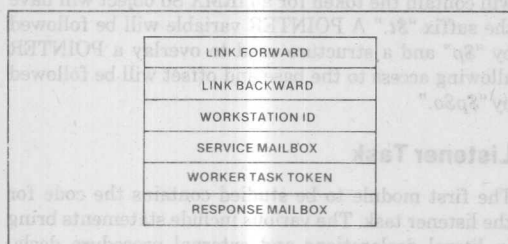


Figure 21. Workstation Descriptor Format

the root job. The token for the root job is obtained by the system call in 1.332.

Whenever a workstation logs on, various actions are taken by the listener task. One of these actions involves adding a descriptor for the workstation to a list so that the state of the workstation can be maintained by the listener task. The list structure is shown in Figure 22. Statements 1.336-1.340 create the root of this list and initialize the list to an empty state.

Line 1.340 marks the beginning of an infinite loop. Most often a task executes a procedure which performs some initialization and then enters an endless loop performing the necessary processing. The literal "forever" translates into "while 1."

A packet is received from the input mailbox by the call in line 1.341. The command field of the message is checked in line 1.343. If the command indicates that a log on request is being made, lines 1.345-1.356 are executed. A log on information segment is created in line 1.345. A mailbox is created to handle further request packets and another is created to be used by the worker task as a response mailbox. The worker

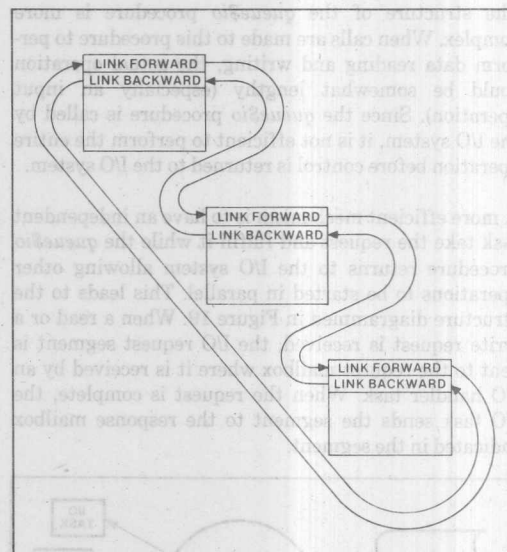


Figure 22. Workstation Descriptor List Structure

task that will handle I/O requests from this workstation is created in line 1.351. Note the use of the structure *data\$seg\$p\$o*, which is declared at the same address as the *POINTER data\$seg\$p*. The *POINTER* is initialized to equal the beginning of the data segment of the worker task module (1.323) and then the base portion is used as a parameter in the create task call.

Once the worker task is created, it will wait at the *log\$on\$info\$mbx* for a segment giving it its initialization information. The segment is sent in line 1.352 and received back as an acknowledgement in line 1.353. At this point, the segment is inserted on the list of active workstation descriptors by the call in line 1.354. Finally the request packet itself is sent to the worker task via the service mailbox for the new worker.

If a log off request is received, lines 1.358 to 1.366 are executed. First, the active workstation list is searched for the ID of the requesting station. If the station is not found to be logged on, the status field is set and the request segment is sent to the workstation through the communications job. If the station is logged on, the descriptor is deleted from the list, the packet is sent along to the worker task, and the descriptor is deleted.

If the command is anything but log on or log off, lines 1.368-1.376 are executed. Once again the station ID is checked to see if it is logged on. If not, an error message is returned. If the station is logged on, the request packet is sent along to the worker task.

WORKER TASK

The code for the worker task is shown in module 2. Upon creation of a new worker task, a segment is received at the *log\$on\$info\$mbx* (2.242). The data in this segment is copied into local variables and the segment is returned (2.247).

The initialization task for this job has already created a user object for this job and has also set up a prefix which points to the root directory for the disk device. These tokens have been cataloged in the root object directory. The worker task obtains these tokens through the sequence of calls 2.248-2.250.

The worker task now enters an infinite loop servicing the workstation it is assigned to. The specific action to be taken by the worker is determined by inspecting the *cmd* field of the request message.

If the command is a log on, the code from 2.256-2.263 is executed. The file name specified in the request segment is attached and opened and thereby made ready for subsequent I/O requests. After this, an acknowledgement is sent back to the workstation via the *output\$request\$mailbox* (2.263).

If a log off command is received, the file is closed and detached, the service and response mailboxes are deleted, a response is sent to the workstation and the worker task is deleted.

If the command is either a read or write command, the operation is performed by calling the I/O system. When the response is received, an acknowledgement is sent to the workstation. Note that the task would normally perform more processing. In this example its duties have been kept simple.

POINTERIZE PROCEDURE

The ASM-86 code for the *pointerize* support routine is shown in Module 3. The token for a segment is the base portion of a 32-bit POINTER to the memory. In order to access the data in a segment, this 16-bit token must be loaded into the base part of a POINTER while the offset portion of the POINTER is set to zero. The base and offset values are returned in the ES and BX registers as specified by the PL/M-86 calling conventions. This is the operation performed by the *pointerize* routine.

LIST MANIPULATION ROUTINES

Lines 4.1-4.47 provide three subroutines used by the tasks in this system to manipulate the list of workstation descriptors. *Insert\$on\$list* (4.15-4.26) inserts the indicated node at the head of the list whose root is given as the first parameter.

Delete\$from\$list (4.27-4.35) unlinks the indicated node from the list it belongs to. *Search\$list* (4.36-4.46) searches a list for the workstation ID given. If the ID is not found, a zero is returned. If the ID is found, the token for that node is returned.

At this point an overview of the configuration process is needed. A more detailed coverage of the process of configuring an iRMX 86 system is provided in the manual entitled "iRMX 86 Configuration Guide for ISIS-II Users."

For each iRMX 86 application, the following steps must be performed.

- 1) Program code for each task in the system must be written and compiled or assembled.
- 2) A memory map for the software must be drawn up.
- 3) The system software must be linked and located.
- 4) The application jobs must be linked and located.
- 5) Tables of configuration data must be drawn up.
- 6) The tabular data from step 5 must be formatted into a memory data block through the use of a set of ASM-86 macros provided with the iRMX 86 product.
- 7) The root job must be linked and located.

The code executed by the root task is part of the iRMX 86 system code. This task is initially the only task in the system. The root task will access the data block constructed by the ASM-86 macros and will create the user jobs specified by the macros. The data for the configuration process for example 1 is shown in Appendix B.

The first page diagrams the memory map for the example. The iterative link and locate process to put these pieces together begins on the second page. The LINK86 and LOC86 commands shown place the iRMX/86 nucleus into memory. The LOCATE map indicates that the last memory location used by the nucleus was 077DFH. Therefore, the next contiguous piece, the I/O system, is located at 077EOH.

This process is repeated for the remainder of the jobs in the system.

When the link and locate process is complete, the information for the ASM-86 macros must be brought together. Worksheets are provided in the iRMX 86 configuration guide to simplify this process.

The filled-out worksheets for the macros are shown in the appendix. A configuration file is constructed using

the editor and the worksheet information is entered into this file. When the file is complete, the configuration table is created by assembling the file CTABLE. A86. This file accesses the configuration file built earlier.

The configuration tables are then linked and located together with the code for the root task and the system generation process is complete.

EXAMPLE 2

INIT\$IIO AND FINISH\$IIO

The *start\$and\$finish* module (5.1-5.371) contains the code for the *init\$534\$Iio* and *finish\$534\$Iio* procedures. The *init\$534\$Iio* procedure creates a segment, shown in Figure 23, which is used to hold the various pieces of information needed by the other driver procedures (5.323). The discussion of this procedure in Chapter 4 pointed out that any errors encountered in the initialization are indicated by the non-zero status and that the assumption is made that any partial creations must be cleaned up by the *init\$Iio* procedure. This assumption is carried out by the check at line 5.324 (and the others at 5.331, 5.335, 5.339 and 5.342).

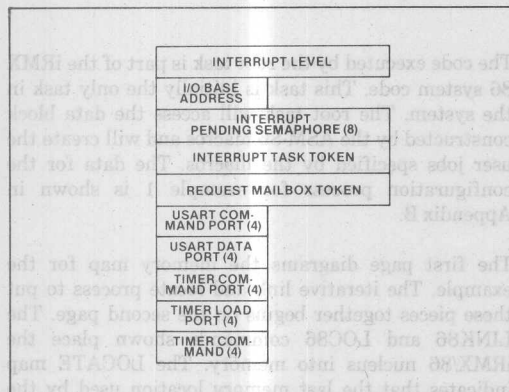


Figure 23. *init\$534\$Iio* Segment Format

The device information contained in the device unit information block for this device is retrieved in line 5.328-5.329. A mailbox to be used for sending I/O request segments to the I/O handler tasks is created in line 5.330. The interrupt task for this job is created by the call in line 5.337.

The *do loop* starting at line 5.340 is executed to create eight semaphores to be used by the interrupt task to indicate the occurrence of an interrupt. Note that the initial value of the semaphore is zero (no interrupt

pending) and the maximum value is one. Since the nature of the 8251 USART device does not support buffering, when a new character overruns the previous character before the interrupt can be serviced, the data is lost. Therefore, there is no need to indicate the occurrence of multiple interrupts pending on the same device.

The call at line 5.345 initializes the programmable devices on the iSBC 534 board. If execution has proceeded to line 5.346, the initialization is complete and a zero status is returned. If an error occurred at any point, the code in lines 5.348-5.356 will clean up the partial initialization.

The *finish\$534\$Iio* procedure (5.358-5.370) undoes the work of the *init\$534\$Iio* procedure. The segment, mailbox, interrupt task and semaphores are all deleted.

The *queue\$534\$Iio* procedure is shown in lines 6.1-6.382. In line 6.322 the function field of the I/O request segment is checked to see if it is within bounds. If it is not, a bad status code is returned. If the function is valid, a *do case* block is executed using the function code as the index.

If a read request is encountered, the auxiliary pointer is set to point to the *ret\$data* structure (initialized earlier by the *init\$534\$Iio* procedure). In line 6.327 the segment is then sent to the request mailbox to be received and processed by an I/O processor task. In lines 6.330-6.334 the same action is taken with write requests.

Since this driver does not support seeking and special functions, the code for these two cases simply returns an error condition.

In the case of an *attach\$device* call, the code in lines 6.341-6.361 is executed. First, two I/O processing tasks are created. All of these tasks execute identical code and each task is capable of servicing a read or a write request on any 8251. Two tasks are created for each 8251 device so that the peak load can always be handled (that is, all receivers and transmitters going simultaneously). Lines 6.346-6.357 perform the initialization of the 8251 USART and the baud rate generators for this channel. The calls in line 6.358 and 6.359 accept an interrupt and a character from the semaphore associated with the receiver just initialized. This is done to clear off an interrupt generated by the 8251 whenever it is initialized.

In the case of a *detach\$device* call, the code in lines 6.363-6.367 sends the I/O request segment to the

request mailbox twice. This is done to signal two of the I/O handler tasks to delete themselves. As discussed earlier in the *attach\$device* section, none of the I/O handler tasks is any different from any of the others. There are two created for each 8251 device which is attached. The protocol set up for their deletion is shown here. When an I/O handler task receives a segment of type "*detach\$device*" it will send the segment to the response mailbox and then delete itself.

The code for the open and close requests is the same. Both cases are supported but are NOPs since no specific action needs to be taken by the driver.

Lines 6.379-6.382 contain the code for the *cancel\$534\$io* procedure. As discussed earlier, this procedure is simply a placeholder and serves no particular purpose.

INTERRUPT CONTROL MODULE

The interrupt handler and interrupt task are shown in lines 7.1-7.329. The interrupt task is the first piece executed. It is created by the *init\$534\$io* procedure. It calls *RQ\$set\$interrupt* in line 7.325 to indicate to the iRMX 86 nucleus that it is an interrupt task.

Once the initialization is complete, the task enters an infinite loop. The call to *RQ\$wait\$interrupt* in line 7.322 causes the task to be put into the asleep state until an interrupt occurrence is signaled. The task will be returned to the READY state when an interrupt occurs, the interrupt handler is started, and the call to *RQ\$signal\$interrupt* is executed at line 7.312. The current interrupt level is then determined by polling the 8259 chip on the iSBC 534 board. Using the encoded level number, a unit is sent to the appropriate semaphore to indicate that an interrupt is pending.

I/O TASK

The final procedure that makes up this driver contains the code for the tasks that perform the actual I/O to the iSBC 534 board. The loop executed by each task starts by waiting at the request mailbox for an I/O request segment. When the segment is sent by the *queue\$534\$IO* procedure, its function code is checked (line 8.327, 8.332, 8.340). If the function is *f\$detach\$device*, the task sends the segment to the response mailbox and then deletes itself.

If the request was for a read, the task fills the buffer with input data. The call at line 8.334 waits for a unit at the semaphore which will indicate a receiver ready on the input line. When the unit is sent by the interrupt task, the character is read in, the pointers and counts are updated, and another unit is requested.

The last request which is recognized by the I/O task is for a write operation. The code for this request is almost identical to the code for a read request. An interrupt from the transmitter is awaited, a character is output and the counts are updated in lines 8.341-8.346.

Once the request is fulfilled, the message is sent to the response exchange in line 8.350.

The configuration of this system is studied next. The code for the iSBC 534 driver is linked directly to the rest of the I/O system libraries. The entry point addresses for the *queue\$534\$io*, *cancel\$534\$io*, *init\$534\$io*, and *finish\$534\$io* procedures are declared in the IOC NFG.A86 file on the I/O system disk. This file also contains the device unit information block (DUIB) structures for the four units on the iSBC 534 board. The unique information for the iSBC 534 device and the units on the device is contained in the device and unit information tables. Pointers to these tables are contained in the DUIB structures. All of this information is shown in Figure 24.

The submit file used to build an I/O system using the iSBC 534 driver is shown in Figure 25. The file DRV534.LIB contains the object files generated by PL/M-86 and ASM-86 from the source code shown in modules 5-9.

SUMMARY

This application note is an introduction to the iRMX 86 Operating System. The requirements of operating systems were studied along with traditional solutions. Following this, the iRMX 86 Operating System was introduced and its correlation with the requirements was studied.

Later in the application note, the topic of system design was covered. Example solutions were studied to solidify a methodology for solving application problems and then the code for these solutions was discussed to gain insight into the details of implementing iRMX 86 systems.

The purpose of a configurable, real-time, multi-purpose operating system is to provide a solid foundation for application software. The iRMX 86 system provides this foundation, giving the software engineer a means to quickly and easily implement new designs. In addition, the iRMX 86 architecture is the bridge to future technology providing the designer with an upgrade path to future hardware and software products.

```

extrn  init534io: near
extrn  queue534io: near
extrn  cancel534io: near
extrn  finish534io: near

; Duib(8): ISBC 534, unit 1
;
; define_duib < '1534.1',
; & 03H,
; & 00033H,
; & 0,
; & 0,0,0,
; & 3,
; & 1,
; & 6,
; & init534io,
; & finish534io,
; & queue534io,
; & cancel534io,
; & dev_534_info,
; & unit_534_1_info
; >
; 534 device info
;
; dev_534_info dw 48H ; level
; db 61 ; priority
; db 040H ; base address
;
; unit info: ISBC 534.0
;
; unit_534_0_info db 4EH ; usart$cmd
; dw 8 ; baud rate
;
; unit info: ISBC 534.1
;
; unit_534_1_info db 4EH ; usart$cmd
; dw 8 ; baud rate
;
; unit info: ISBC 534.2
;
; unit_534_2_info db 4EH ; usart$cmd
; dw 8 ; baud rate
;
; unit info: ISBC 534.3
;
; unit_534_3_info db 4EH ; usart$cmd
; dw 8 ; baud rate

```

Figure 24. IOCNGF A86 File Entries for ISBC 534™ Driver

```

; ios(date,origin)
; Sample I/O System .csd file to link and locate an I/O System.
; This file links an I/O System with the timer included.
; This .csd file assumes the I/O System configuration module is
; iocnfg.a86 (found on the release diskette).
; The origin parameter sets the low address of the I/O System;
; all the segments are contiguous in memory.
asm86 :fl:iocnfg.a86 date(%0) print(:f5:iocnfg.lst)
link86 &
:fl:ios.lib(ioinit), &
:fl:iocnfg.obj, &
:fl:ios.lib, &
:fl:drv534.lib, &
:fl:rpifc.lib, &
to :fl:ios.lnk map print(:fl:ios.mp1)
loc86 :fl:ios.lnk to :fl:ios map sc(3) print(:fl:ios.mp2) &
oc(noli,nopl,nocm,nosb) &
order(classes(code,data,stack,memory)) &
addresses(classes(code(%1))) &
segsize(stack(0))

```

Figure 25. Submit File for Generating an I/O System with the ISBC 534™ Driver

APPENDIX A	2-97
APPENDIX B	2-121

APPENDIX A
Code Listings

APPENDIX A
Code Listings

Module 1

ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE LISTENERMODULE
 OBJECT MODULE PLACED IN :F1:listen.OBJ
 COMPILER INVOKED BY: plm86 :F1:listen.plm PRINT(:F1:LISTEN.LST)
 DEBUG COMPACT OPTIMIZE(3) ROM DATE(5/28/80)

```

1      listener$module:
      do;

/*****

LISTENER: TASK.

This task creates segments, sends them to the input service
job to be filled with input packet info. Upon response
the info is checked to see what action needs to be taken.
If a log$on request is sensed, a worker task, service
mailbox, and response mailbox are created and the packet is
sent along to the worker task. If a log$off is sensed all
local reference to the workstation is deleted and the packet
is sent along to tell the worker to delete himself. If an
I/O request is sensed the station ID is checked to make
sure it is logged on. If it is, the packet is sent along to
the worker. If it isn't an error packet is sent back to the
requesting workstation.

*****/

$include(:f2:common.lit)
SSAVE NOLIST
$include(:f1:node.lit)
/* literal declaration of node descriptor for list utilities */
11 1  declare
      node literally 'structure(
      link$ word,
      link$b word,
      work$station$ID word,
      service$mbx$st word,
      worker$task$st word,
      resp$mbx$st word)';
$include(:f1:lstutl.ext)
/* external declarations for list manipulation utilities */
SSave nolist
$include(:f1:pointtr.ext)
/* external declaration of pointerize procedure */
SSave nolist
$include(:f1:rqpckt.lit)
/* literal declaration for request packet structure */
24 1  declare req$segment$st struct literally 'structure(
      funct word,
      count word,
      actual word,
      ex$val word,
      work$station$ID word,
      cmd word,
      share word,
      mode word,
      status word,
      file$name (64) byte,
      buf (128) byte)';
$include(:f2:nucprm.ext)
SSAVE NOLIST

321 1  worker$task: procedure external;
322 2  end worker$task;

```

Module 1, continued

```

323 1 declare
begin$listener$task$data byte public,
begin$worker$task$data byte external,
log$on$info$mbox$token public,
ex$val word,
log$on$mbox$name (7) byte data(6,'LOG$ON'),
packet$size literally '132',
f$read literally '5',
f$write literally '6',
log$on literally '0',
log$off literally '1',
not$logged$on literally '1',
(root$job$token,input$request$mbox$token,
(output$request$mbox$token,resp$mbox$token),
(work$station$list$root$token,req$segment$token),
(log$on$info$seg$token,dummy$token,ws$desc$token),
(req$segment$token,work$station$list$root$token) pointer,
(log$on$info$seg$token,data$seg$token,ws$desc$token) pointer,
(req$segment based req$segment$token) req$segment$struct,
(work$station$list$root based work$station$list$root$token) node,
(log$on$info$seg based log$on$info$seg$token) node,
data$seg$token so structure(offset word, base word) at(@data$seg$token),
(ws$desc based ws$desc$token) node;

324 1 return$error$to$WS: procedure;

325 2 req$segment.funct=f$write;
326 2 req$segment.status=not$logged$on;
327 2 call rq$send$message(output$request$mbox$token,req$segment$token,0,@ex$val);
328 2 return;
329 2 end;

330 1 Listener: procedure public; /* task */

331 2 log$on$info$mbox$token=rq$create$mailbox(0,@ex$val);
332 2 root$job$token=rq$get$task$tokens(3,@ex$val);
333 2 input$request$mbox$token=rq$lookup$object(
/* job */ root$job$token,
/* name */ @('INPUT$REQ'),
/* time limit */ 0FFFFH,
/* status ptr */ @ex$val);

334 2 output$request$mbox$token=rq$lookup$object(
/* job */ root$job$token,
/* name */ @('OUTPUT$REQ'),
/* time limit */ 0FFFFH,
/* status ptr */ @ex$val);

335 2 resp$mbox$token=rq$create$mailbox(0,@ex$val);
336 2 work$station$list$root$token=rq$create$segment(16,@ex$val);
337 2 work$station$list$root$token=pointerize(work$station$list$root$token);
338 2 work$station$list$root$token.link$fb=work$station$list$root$token;
339 2 work$station$list$root$token.workstation$ID=0;

340 2 do forever;

341 3 req$segment$token = rq$receive$message(
/* mbox token */ input$request$mbox$token,
/* time limit */ 0FFFFH,
/* response ptr */ @dummy$token,
/* status ptr */ @ex$val);
342 3 req$segment$token=pointerize(req$segment$token);

343 3 if req$segment.cmd= log$on then
344 3 do;

```

Module 1, continued

```

345 4      log$on$info$seg$st=rq$create$segment(
          /* size */ 16,
          /* status ptr */ @ex$val);
346 4      log$on$info$seg$sp=pointerize(
          log$on$info$seg$st);
347 4      log$on$info$seg.service$mbox$st=
          rq$create$mailbox(0,@ex$val);
348 4      log$on$info$seg.resp$mbox$st=
          rq$create$mailbox(0,@ex$val);
349 4      log$on$info$seg.work$station$ID=
          req$segment.work$station$ID;
350 4      data$seg$sp=@begin$worker$task$data;
351 4      log$on$info$seg.worker$task$st=
          rq$create$task(
          /* priority */      200,
          /* start addr */    @worker$task,
          /* data seg ptr */  data$seg$sp$.base,
          /* stack pointer */ 0,
          /* stack size */    500,
          /* task flags */    0,
          /* status ptr */    @ex$val);

352 4      call rq$send$message(
          /* mbox token */    log$on$info$mbox$st,
          /* object token */  log$on$info$seg$st,
          /* response token */ resp$mbox$st,
          /* status ptr */    @ex$val);

353 4      log$on$info$seg$st=rq$receive$message(
          /* mailbox token */  resp$mbox$st,
          /* time limit */     0FFFFH,
          /* response token */ @dummy$st,
          /* status ptr */     @ex$val);

354 4      call insert$on$list(work$station$list$root$st,
          log$on$info$seg$st);
355 4      call rq$send$message(
          /* mbox tok */    log$on$info$seg.service$mbox$st,
          /* obj tok */     req$segment$st,
          /* response */    0,
          /* status */      @ex$val);

356 4      end;

357 3      else if req$segment.cmd = log$off then
358 3      do;
359 4          ws$desc$st=search$list(work$station$list$root$st,
          req$segment.work$station$ID);
360 4          if ws$desc$st = 0 then
361 4              call return$error$to$WS;
          else
362 4              do;
363 5                  ws$desc$st=pointerize(ws$desc$st);
364 5                  call delete$from$list(
          ws$desc$st);
365 5                  call rq$send$message(
          ws$desc.service$mbox$st,
          req$segment$st,
          0,
          @ex$val);

366 5              end;

367 4      end;

          else
368 3      do;
369 4          ws$desc$st=search$list(work$station$list$root$st,
          req$segment.work$station$ID);

```


Module 1, continued

```

370 4      if ws$desc$t=0 then;
371 4          call return$error$to$WS;
          else
372 4      do;
373 5          ws$descp=pointerize(ws$desc$t);
374 5          call rq$send$message(
              ws$desc.service$mbox$t,
              req$segment$t,
              0,
              @ex$val);
375 5      end;
376 4      end;
377 3      call rq$delete$segment(req$segment$t,@ex$val);
378 3      end; /* of do forever */

379 2      end; /* of listener task */

380 1      end listener$module;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0281H      641D
CONSTANT AREA SIZE = 0000H      0D
VARIABLE AREA SIZE = 002BH      43D
MAXIMUM STACK SIZE = 0018H      24D
694 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

Module 2

ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE WORKERTASK
 OBJECT MODULE PLACED IN :F1:worker.OBJ
 COMPILER INVOKED BY: plm86 :F1:worker.plm PRINT(:F1:WORKER.LST)
 DEBUG COMPACT OPTIMIZE(3) ROM DATE(5/28/80)

```

1      worker$task:
      do;

/*****

WORKER$TASK: TASK.

This module contains the code executed by the worker tasks.
When started, the task goes to a mailbox to receive a segment
containing initialization information. Using this information
the task services a service mailbox performing any I/O functions
requested of it. When a log$off request comes in the worker
task closes and detaches the file and deletes itself.

*****/

      $include(:f1:nucprm.ext)
=      $$SAVE NOLIST
      $include(:f1:iosys.ext)
=      $$save nolist
      $include(:f1:node.lit)
=      /* literal declaration of node descriptor for list utilities */
=      $$save nolist
      $include(:f2:common.lit)
=      $$SAVE NOLIST
      $include(:f1:pointtr.ext)
=      /* external declaration of pointerize procedure */
=      $$save nolist
      $include(:f1:rqpkt.lit)
=      /* literal declaration for request packet structure */
=      $$save nolist

239 1      declare
          read literally '1',
          write literally '5',
          log$on literally '2',
          log$off literally '3',
          (log$on$info$mbox$st,output$request$mbox$st) token external;

240 1      worker$task: procedure reentrant public;

241 2      declare
          (log$on$info$seg$st,log$on$resp$mbox$st,resp$mbox$st,
          root$job$st,user$object$st,prefix$st,iors$st,
          service$mbox$st,conn$st,req$seg$st) token,
          (log$on$info$sp,req$seg$sp) pointer,
          (req$seg based req$seg$sp) req$segment$struc,
          (log$on$info based log$on$info$sp) node,
          (dummy$st,ex$val,work$station$ID) word;

242 2      log$on$info$seg$st=rq$receive$message(
          /* mbox token */    log$on$info$mbox$st,
          /* time limit */    0FFFFH,
          /* response ptr */  @log$on$resp$mbox$st,
          /* status ptr */    @ex$val);

243 2      log$on$info$sp=pointerize(log$on$info$seg$st);
244 2      service$mbox$st=log$on$info.service$mbox$st;
245 2      resp$mbox$st=log$on$info.resp$mbox$st;
246 2      work$station$ID=log$on$info.work$station$ID;

```

Module 2, continued

```

247 2      call rq$sendMessage(
          /* mbox token */ log$on$resp$mbox$st,
          /* object token */ log$on$info$seg$st,
          /* response token */ 0,
          /* status ptr */ @ex$val);

248 2      root$job$st=rq$get$task$tokens(3,@ex$val);
249 2      user$object$st=rq$lookup$object(
          /* job token */ root$job$st,
          /* name */ @ (11,'USER$OBJECT'),
          /* time limit */ 0FFFFH,
          /* status ptr */ @ex$val);
250 2      prefix$st=rq$lookup$object(
          /* job token */ root$job$st,
          /* name */ @ (6,'PREFIX'),
          /* time limit */ 0FFFFH,
          /* status ptr */ @ex$val);

251 2      do forever;

252 3          req$seg$st=rq$receive$message(
          /* mailbox token */ service$mbox$st,
          /* time limit */ 0FFFFH,
          /* response ptr */ @dummy$st,
          /* status ptr */ @ex$val);

253 3          req$seg$sp=pointerize(req$seg$st);

254 3          if req$seg.cmd=log$on then
255 3              do;
256 4              call rq$a$attach$file(
          /* user object */ user$object$st,
          /* prefix token */ prefix$st,
          /* pathname */ @req$seg.file$name,
          /* resp token */ resp$mbox$st,
          /* status ptr */ @ex$val);
257 4              iors$st=rq$receive$message(
          /* mbox token */ resp$mbox$st,
          /* time limit */ 0FFFFH,
          /* resp ptr */ @dummy$st,
          /* status ptr */ @ex$val);
258 4              call rq$delete$segment(iors$st,@ex$val);
259 4              call rq$a$open(
          /* connection */ conn$st,
          /* mode */ req$seg.mode,
          /* share */ req$seg.share,
          /* resp token */ resp$mbox$st,
          /* status ptr */ @ex$val);
260 4              iors$st=rq$receive$message(
          /* mbox token */ resp$mbox$st,
          /* time limit */ 0FFFFH,
          /* resp ptr */ @dummy$st,
          /* status ptr */ @ex$val);
261 4              call rq$delete$segment(iors$st,@ex$val);
262 4              req$seg.status=0;
263 4              call rq$sendMessage(
          /* mbox token */ output$request$mbox$st,
          /* object token */ req$seg$st,
          /* resp ptr */ 0,
          /* status ptr */ @ex$val);

264 4              end;

265 3          else if req$seg.cmd=log$off then
266 3              do;
267 4              call rq$a$close(
          /* connection */ conn$st,
          /* resp token */ resp$mbox$st,
          /* status ptr */ @ex$val);

```

Module 2, continued

```

268 4      iorsSt= rq$receive$message(
          /* mbox token */      resp$mboxSt,
          /* time limit */      0FFFFH,
          /* resp ptr */        @dummySt,
          /* status ptr */      @ex$val);
269 4      call rq$delete$segment(iorsSt,@ex$val);
270 4      call rq$a$delete$connection(
          /* connection */      connSt,
          /* response ptr */    resp$mboxSt,
          /* status ptr */      @ex$val);
271 4      iorsSt=rq$receive$message(
          /* mbox token */      resp$mboxSt,
          /* time limit */      0FFFFH,
          /* response ptr */    @dummySt,
          /* status ptr */      @ex$val);
272 4      call rq$delete$segment(iorsSt,@ex$val);
273 4      call rq$delete$mailbox(service$mboxSt,@ex$val);
274 4      call rq$delete$mailbox(resp$mboxSt,@ex$val);
275 4      req$seg.status=0;
276 4      call rq$send$message(
          /* mbox token */      output$request$mboxSt,
          /* object token */    req$segSt,
          /* resp token */      0,
          /* status ptr */      @ex$val);
277 4      call rq$delete$task(0,@ex$val);
278 4      end;

279 3      else if req$seg.cmd=read then
280 3      do;

281 4          call rq$a$read(
              /* connection */      connSt,
              /* buf ptr */          @req$seg.buf,
              /* count */            req$seg.count,
              /* resp token */      resp$mboxSt,
              /* status ptr */      @ex$val);
282 4      iorsSt=rq$receive$message(
          /* mbox token */      resp$mboxSt,
          /* time limit */      0FFFFH,
          /* resp ptr */        @dummySt,
          /* status ptr */      @ex$val);
283 4      call rq$delete$segment(iorsSt,@ex$val);
284 4      req$seg.status=0;
285 4      call rq$send$message(
          /* mbox token */      output$request$mboxSt,
          /* object token */    req$segSt,
          /* resp token */      0,
          /* status ptr */      @ex$val);
286 4      end;

287 3      else if req$seg.cmd=write then
288 3      do;

289 4          call rq$a$write(
              /* connection */      connSt,
              /* buf ptr */          @req$seg.buf,
              /* count */            req$seg.count,
              /* resp token */      resp$mboxSt,
              /* status ptr */      @ex$val);
290 4      iorsSt=rq$receive$message(
          /* mbox token */      resp$mboxSt,
          /* time limit */      0FFFFH,
          /* resp ptr */        @dummySt,
          /* status ptr */      @ex$val);
291 4      call rq$delete$segment(iorsSt,@ex$val);

```


Module 2, continued

```

292  4      call rq$send$message(
          /* mbox token */ output$request$mbox$st,
          /* object token */ req$seg$st,
          /* resp token */ 0,
          /* status ptr */ @ex$val);
293  4      end;
          end; /* of do forever */
295  2      end; /* of task */
296  1      end worker$task;

```

MODULE INFORMATION:

```

CODE AREA SIZE = 0288H 648D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 0000H 0D
MAXIMUM STACK SIZE = 0034H 52D
717 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

Module 3

ISIS-II MCS-86 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE POINTR
 OBJECT MODULE PLACED IN :F1:POINTR.OBJ
 ASSEMBLER INVOKED BY: asm86 :f1:pointr.a86 debug pr(:f5:pointr.lst)

LOC	OBJ	LINE	SOURCE
		1	\$title(pointerize Utility)
0004		2	arg_off equ 4 ; set args for "DELUXE"
----		3	
----		4	code segment word public 'CODE'
----		5	code ends
		6	
----		7	cgroup group code
		8	code segment
		9	assume cs:cgroup
		10	
0000		11	pointerize proc near
		12	public pointerize
0000 55		13	push bp ; save
0001 8BEC		14	mov bp, sp ; mark stack
		15	
0004 []		16	token equ word ptr [bp + arg_off + 0]
		17	
0003 8E4604		18	mov es, token ; get base
0006 33DB		19	xor bx, bx ; zap offset
		20	
		21	; mov sp, bp ; restore stack
0008 5D		22	pop bp
0009 C20200		23	ret 2
		24	pointerize endp
----		25	code ends
		26	end

ASSEMBLY COMPLETE, NO ERRORS FOUND

Module 4

ISIS-II PL/M-86 X167 COMPILATION OF MODULE LISTUTILITIESMODULE
 OBJECT MODULE PLACED IN :F1:lstutl.OBJ
 COMPILER INVOKED BY: plm86 :F1:lstutl.plm PRINT(:F5:LSTUTL.LST)
 DEBUG COMPACT OPTIMIZE(3) ROM DATE(3/7/80)

```

1      list$utilities$module:
      do;

      /*****
      LIST$UTILITIES: PUBLIC PROCEDURES.

      This module contains three list manipulation utilities.
      Insert$on$list takes the given node and inserts it on the
      list indicated by the root node parameter. Delete$from
      list unlinks the indicated node from the list it is
      linked to. Search$list scans the list from the root looking
      for the indicated node. If found, the token for the node
      is returned. If not found, a zero is returned.

      *****/

      $include(:f4:common.lit)
      = $SAVE NOLIST
      $include(:f1:node.lit)
      = /* literal declaration of node descriptor for list utilities */
      = $save nolist
      $include(:f1:pointer.ext)
      = /* external declaration of pointerize procedure */
      = $save nolist

15 1      Insert$on$list: procedure( root$token,new$desc$token) reentrant public;
16 2      declare
          (root$token,new$desc$token,fwd$desc$token) token,
          (root$ptr,new$desc$ptr,fwd$desc$ptr) pointer,
          (root based root$ptr) node,
          (new$desc based new$desc$ptr) node,
          (fwd$desc based fwd$desc$ptr) node;

17 2      root$ptr=pointerize(root$token);
18 2      new$desc$ptr=pointerize(new$desc$token);
19 2      fwd$desc$token=root.link$f;
20 2      fwd$desc$ptr=pointerize(fwd$desc$token);
21 2      root.link$f=new$desc$token;
22 2      new$desc.link$f=fwd$desc$token;
23 2      new$desc.link$b=root$token;
24 2      fwd$desc.link$b=new$desc$token;
25 2      return;

26 2      end; /* insert$on$list */

27 1      Delete$from$list: procedure(desc$token) reentrant public;
28 2      declare
          desc$token,
          (desc$ptr,b$desc$ptr,f$desc$ptr) pointer,
          (desc based desc$ptr) node,
          (b$desc based b$desc$ptr) node,
          (f$desc based f$desc$ptr) node;

29 2      desc$ptr=pointerize(desc$token);
30 2      b$desc$ptr=pointerize(desc.link$b);
31 2      f$desc$ptr=pointerize(desc.link$f);
32 2      b$desc.link$f=desc.link$f;
33 2      f$desc.link$b=desc.link$b;
34 2      return;

```

Module 4, continued

```

35  2      end; /* delete$from$list */
36  1      search$list: procedure(root$st,WSSID) word reentrant public;
37  2      declare
          (root$st,WSSID) word,
          (s$desc$p,root$p) pointer,
          (root based root$p) node,
          (s$desc based s$desc$p) node,
          s$desc$p$o structure (offset word, base word) at(@s$desc$p),
          temp pointer;

38  2      s$desc$p=pointerize(root$st);
39  2      next$node:
          if s$desc.work$station$ID=WSSID then
40  2          return s$desc$p$o.base;
41  2          if s$desc.link$f = root$st then
42  2              return 0;
43  2          temp=pointerize(s$desc.link$f);
44  2          s$desc$p=temp;
45  2          goto next$node;

46  2      end; /* search$list */
47  1      end list$utilities$module;

```

MODULE INFORMATION:

CODE AREA SIZE	= 00FEH	254D
CONSTANT AREA SIZE	= 0000H	0D
VARIABLE AREA SIZE	= 0000H	0D
MAXIMUM STACK SIZE	= 0018H	24D
114 LINES READ		
0 PROGRAM ERROR(S)		

END OF PL/M-86 COMPILATION

Module 5

ISIS-II PL/M-86 X167 COMPILATION OF MODULE STARTANDFINISH
 OBJECT MODULE PLACED IN :F1:strfin.OBJ
 COMPILER INVOKED BY: plm86 :F1:strfin.plm PRINT(:F5:STRFIN.LST)
 DEBUG COMPACT OPTIMIZE(2) ROM DATE(4/28/80)

```

1      start$and$finish:
      do;

/*****

      INIT$534$IO and FINISH$534$IO: PUBLIC PROCEDURES.

      This module contains the init$534$IO and the FINISH$534$IO
      procedures which can be called by the RMX/86 I/O system. START$IO
      is called just before the first attach$device is performed.
      It will create the interrupt task and the eight interrupt$pending
      semaphores. The FINISH$IO procedure is called just after the
      last detach$device is performed. It undoes everything the START$IO
      call did.

      *****/

      $include(:f4:nucprm.ext)
=      $$SAVE NOLIST
      $include(:f4:common.lit)
=      $$SAVE NOLIST
      $include(:f1:duib.lit)
=      /* duib structure definition */
=      $$save nolist
      $include(:f4:nerror.lit)
=
=      $$SAVE NOLIST
      $include(:f1:pointerize.ext)
=      /* external declaration of pointerize procedure */
=      $$save nolist
      $include(:f1:ret$tda.lit)
=      /* literal declaration of ret$data structure for init$534$io */
=      $$save nolist

314  1      init$534$hw: procedure(data$p) external;
315  2          declare data$p pointer;
316  2      end init$534$hw; /* initializes 534 hardware */

317  1      int$534$task: procedure external;
318  2      end int$534$task;

319  1      declare
          begin$int$534$data byte external,
          IO$base$addr byte public,
          int$level.word public,
          g$ret$data$p pointer public,
          req$mbox$token public;

320  1      init$534$IO: procedure(duib$p,ret$data$token,status$p) reentrant public;
321  2      declare
          (duib$p,ret$data$token,status$p) pointer,
          (duib based duib$p) dev$unit$info$block,
          (ret$data$token based ret$data$token) token,
          (status based status$p) word,
          dev$info$p pointer,
          dev$info based dev$info$p structure(
              level word,
              priority byte,
              IO$base$addr byte),

```

Module 5, continued

```

ex$val word,
data$seg$po pointer,
data$seg$po structure(offset word,base word) at(@data$seg$po),
(i,j) byte;

322 2 declare
      ret$data$po pointer,
      ret$data based ret$data$po structure(ret$data$struct);

323 2 ret$data$=rq$create$segment(size(ret$data),@ex$val);
324 2 if ex$val <> 0 then
325 2   goto err0;
326 2 g$ret$data$po,ret$data$po=pointerize(ret$data$);
327 2 dev$info$po=duib.dev$info$po;
328 2 IO$base$addr,ret$data.IO$base=dev$info.IO$base$addr;
329 2 int$level,ret$data.int$level=dev$info.level;

/* create the request mailbox */

330 2 ret$data.request$mbx$st,req$mbx$st
      =rq$create$mailbox(0,@ex$val);
331 2 if ex$val <> 0 then
332 2   goto err1;
333 2 ret$data.resp$mbx$st=rq$create$mailbox(0,@ex$val);
334 2 if ex$val <> 0 then
335 2   goto err2; /* clean up partial creation */

336 2 data$seg$po=@begin$int$534$data;
337 2 ret$data.int$task$st=rq$create$task(
      /* priority */ dev$info.priority,
      /* entry point */ @int$534$task,
      /* data segment */ data$seg$po.base,
      /* stack pointer */ 0,
      /* stack size */ 400,
      /* task flags */ 0,
      /* status pointer */ @ex$val);
338 2 if ex$val <> 0 then
339 2   goto err3; /* can't create. clean up partial creation */

340 2 do i=0 to 7; /* create semaphores */
341 3   ret$data.int$sma(i)=rq$create$semaphore(
      /* initial value */ 0,
      /* max value */ 1,
      /* priority queue */ 1,
      /* status ptr */ @ex$val);

342 3   if ex$val <> 0 then

343 3     goto err4; /* clean up partial creation */

344 3   end;
345 2 call init$534$hwr(ret$data$po);
346 2 status=ESOK;
347 2 return;

348 2 err4:
      do j=0 to i;
349 3   call rq$delete$semaphore(ret$data.int$sma(j),status$po);
350 3   end;
351 2 call rq$reset$interrupt(dev$info.level,status$po);
352 2 err3:
      call rq$delete$mailbox(ret$data.resp$mbx$st,status$po);
353 2 err2:
      call rq$delete$mailbox(ret$data.request$mbx$st,status$po);
354 2 err1:
      call rq$delete$segment(ret$data$st,status$po);

```

Module 5, continued

```

355 2      err0:
356 2          status=ex$val; /* restore original status condition */
357 2      end; /* of procedure */

358 1      finish$534$I0: procedure(duib$sp,ret$data$st) reentrant public;
359 2          declare
              duib$sp pointer,
              dev$info$sp pointer,
              dev$info based dev$info$sp structure(
                  level word,
                  priority byte,
                  IOSbase$addr byte),
              ret$data$sp pointer,
              ret$data based ret$data$sp structure(ret$data$st),
              (duib based duib$sp) dev$unit$info$block,
              ret$data$st token,
              i byte,
              ex$val word;

360 2          dev$info$sp=duib.dev$info$sp;
361 2          ret$data$sp=pointerize(ret$data$st);
362 2          call rq$reset$interrupt(dev$info.level,@ex$val);
363 2          call rq$delete$mailbox(ret$data.request$mbx$st,@ex$val);
364 2          call rq$delete$mailbox(ret$data.resp$mbx$st,@ex$val);
365 2          do i=0 to 7;
366 3              call rq$delete$semaphore(
                  ret$data.int$sema(i),
                  @ex$val);
367 3          end;
368 2          call rq$delete$segment(ret$data$st,@ex$val);
369 2          return;
370 2      end; /* of procedure */
371 1      end start$and$finish;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0220H      544D
CONSTANT AREA SIZE  = 0000H       0D
VARIABLE AREA SIZE  = 0009H       9D
MAXIMUM STACK SIZE  = 0034H      52D
671 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

Module 6

ISIS-II PL/M-86 X167 COMPILATION OF MODULE QUEUE534IOMODULE
 OBJECT MODULE PLACED IN :F1:queio.OBJ
 COMPILER INVOKED BY: plm86 :F1:queio.plm PRINT(:F5:QUEIO.LST)
 DEBUG COMPACT OPTIMIZE(2) ROM DATE(4/25/80)

```

1      queue$534$io$module:
      do;

/*****

      QUEUE$534$IO. PUBLIC PROCEDURE.

      This procedure is called by the I/O System to queue
      an I/O request to the 534 board. The function field
      in the IORS is used to determine what specific action
      to take. Module also contains a dummy cancel$534$io
      procedure.

      *****/

      $include(:f4:nucprm.ext)
=      $SAVE NOLIST
      $include(:f4:common.lit)
=      $SAVE NOLIST
      $include(:f4:nerror.lit)
=
=      $SAVE NOLIST
      $include(:f1:pintr.ext)
=      /* external declaration of pointerize procedure */
=      $save nolist
      $include(:f1:duib.lit)
=      /* duib structure definition */
=      $save nolist
      $include(:f1:iors.lit)
=      /* literal declaration for iors */
=      $save nolist
      $include(:f1:ret$dt$lit)
=      /* literal declaration of ret$data structure for init$534$io */
=      $save nolist

315 1      io$534$task: procedure external;
316 2      end io$534$task;

317 1      declare
      begin$io$task$data byte external;

318 1      queue$534$io: procedure(iors$st,duib$sp,ret$data$st) reentrant public;
319 2      declare
      (iors$st,ret$data$st) token,
      data$seg$sp pointer,
      data$seg$sp$so structure(offset word,base word) at(@data$seg$sp),
      IDDR literally '2AH',
      (duib$sp,ret$data$sp,iors$sp) pointer,
      (duib based duib$sp) dev$unit$info$block,
      (ret$data based ret$data$sp) structure(ret$data$st$truc),
      (iors based iors$sp) IO$request$result$segment,
      io$task$st token,
      unit$info$sp pointer,
      unit$info based unit$info$sp structure(
      usart$cmd byte,
      baud$rate word),
      i byte,
      dummy$st token,
      ex$val word;

```


Module 6, continued

```

320 2      iors$P=pointerize(iors$t);
321 2      ret$data$P=pointerize(ret$data$t);
322 2      if iors.funcnt > 7 then
323 2          goto bad$request;
324 2      do case iors.funcnt;
325 3          do; /* case 0-- read */
326 4              iors.aux$P=ret$data$P;
327 4              call rq$send$message(
                  /* mbox */ ret$data.request$mbox$t,
                  /* token */ iors$t,
                  /* resp */ 0,
                  /* status ptr */ @ex$val);
328 4              return;
329 4          end;
330 3      do; /* case 1-- write */
331 4          iors.aux$P=ret$data$P;
332 4          call rq$send$message(
                  /* mbox */ ret$data.request$mbox$t,
                  /* token */ iors$t,
                  /* resp */ 0,
                  /* status ptr */ @ex$val);
333 4          return;
334 4          end;
335 3      do; /* case 2--seek (illegal) */
336 4          goto bad$request;
337 4          end;
338 3      do; /* case 3-- special (illegal) */
339 4          goto bad$request;
340 4          end;
341 3      do; /* case 4-- attach$device */
          /* create two I/O tasks */
342 4          data$seg$P=@begin$IO$task$data;
343 4          do i=0 to 1;
344 5              io$task$t= rq$create$task(
                  /* priority */ 150,
                  /* entry pnt */ @io$534$task,
                  /* data seg */ data$seg$P$.base,
                  /* stack ptr */ 0,
                  /* stack size */ 500,
                  /* task flags */ 0,
                  /* status ptr */ @ex$val);
345 5              end;
346 4          unit$info$P=duib.unit$info$P;
347 4          do i=0 to 3;
348 5              output(ret$data.usart$cmd$port(iors.unit))=0;
349 5              end;
350 4          output(ret$data.usart$cmd$port(iors.unit))=40H;
351 4          output(ret$data.usart$cmd$port(iors.unit))=
              unit$info.usart$cmd;
352 4          output(ret$data.usart$cmd$port(iors.unit))=27H;
353 4          output(ret$data.io$base+0CH)=0; /* select cntrl blk */
354 4          output(ret$data.timer$cmd$port(iors.unit))=
              ret$data.timer$cmd(iors.unit);
355 4          output(ret$data.timer$load$port(iors.unit))=
              low(unit$info.baud$rate);
356 4          output(ret$data.timer$load$port(iors.unit))=
              high(unit$info.baud$rate);

```

Module 6, continued

```

357 4      output(ret$data.IO$base+0DH)=0; /* select data blk */
      /* accept interrupt and character from receiver */

358 4      dummy$=rq$receive$units(
      /* sema */ ret$data.int$sema( 2 * iors.unit),
      /* units */ 1,
      /* time$out */ 0,
      /* status */ @ex$val);
359 4      i=input(ret$data.usart$data$port( iors.unit ));
360 4      goto ok$send$resp;
361 4      end;

362 3      do; /* case 5-- detach$device */

/* send two copies of the detach request to the request mailbox.
   This will signal to two of the I/O tasks that they are to
   delete themselves */

363 4      call rq$send$message(
      /* mbox token */ ret$data.request$mbox$st,
      /* object token */ iors$st,
      /* response */ ret$data.resp$mbox$st,
      /* status */ @ex$val);
364 4      dummy$=rq$receive$message(
      /* mbox token */ ret$data.resp$mbox$st,
      /* time$limit */ 0FFFFH,
      /* response ptr */ @dummy$,
      /* status ptr */ @ex$val);
365 4      call rq$send$message(
      /* mbox token */ ret$data.request$mbox$st,
      /* object token */ iors$st,
      /* response */ ret$data.resp$mbox$st,
      /* status */ @ex$val);
366 4      dummy$=rq$receive$message(
      /* mbox token */ ret$data.resp$mbox$st,
      /* time$limit */ 0FFFFH,
      /* response ptr */ @dummy$,
      /* status ptr */ @ex$val);
367 4      goto ok$send$resp;
368 4      end;

369 3      do; /* case 6-- open */
370 4      goto ok$send$resp;
371 4      end;

372 3      do; /* case 7-- close */
373 4      goto ok$send$resp;
374 4      end;
375 3      end; /* do case */
376 2      return;
377 2      bad$request:
      iors.status=IDDR;
      goto send$resp;
378 2      ok$send$resp:
379 2      iors.status=ESOK;
      send$resp:
      call rq$send$message(iors.resp$mbox, iors$st, 0, @ex$val);
381 2      return;
382 2      end; /* procedure */

383 1      cancel$534$io: procedure(iors$st, duib$p, ret$data$st) public;
384 2      declare
      (iors$st, ret$data$st) token,
      duib$p pointer;

385 2      return;

```

Module 0, continued

```

386 2 // end;
387 1 end queue$534$io$module;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 020CH      524D
CONSTANT AREA SIZE  = 0000H      0D
VARIABLE AREA SIZE  = 0000H      0D
MAXIMUM STACK SIZE  = 0038H      56D
729 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE INTERRUPT534MODULE
 OBJECT MODULE PLACED IN :F1:int534.OBJ
 COMPILER INVOKED BY: plm86 :F1:int534.plm PRINT(:F1:INT534.LST)
 DEBUG COMPACT OPTIMIZE(2) ROM DATE(5/28/80)

```

    $nointvector
1      Interrupt$534$module:
        do;

/*****

    INT$534$TASK and INT$534$HND:
    PUBLIC PROCEDURES:

    This module contains the interrupt handler and the interrupt
    task for the 534 board interrupt. The handler simply calls
    signal$interrupt and the task reads the ISR on the 534
    board's 8259 and sends a unit to one of eight interrupt$
    pending semaphores to signal the occurrence of the event.

    *****/

    $include(:f2:nucprm.ext)
    = $SAVE NOLIST
    $include(:f1:ret$da.lit)
    = /* literal declaration of ret$data structure for init$534$io */
    = $save nolist
    $include(:f2:common.lit)
    = $SAVE NOLIST

308 1      declare
            begin$int$534$data byte public,
            g$ret$data$pointer external,
            IO$base$addr byte external,
            int$level word external;

309 1      int$534$hnd: procedure interrupt 5;

310 2      declare
            1 word,
            ex$val word;

311 2      l=rq$get$level(@ex$val);
312 2      call rq$signal$interrupt(1,@ex$val);
313 2      return;
314 2      end;

315 1      int$534$task: procedure reentrant public;

316 2      declare
            IO$534$base byte,
            int$534$level word,
            ret$data$pointer,
            ret$data based ret$data$pointer structure(ret$data$struct),
            c$level byte,
            ex$val word,
            eoi literally '20H';

317 2      IO$534$base=IO$base$addr;
318 2      int$534$level=int$level;
319 2      ret$data$pointer=g$ret$data$pointer;
320 2      call rq$set$interrupt(
            /* level */      int$534$level,
            /* flags */      1,
            /* entry point */ interrupt$ptr(int$534$hnd),
            /* data segment */ 0,
            /* status ptr */  @ex$val);

```


Module 7, continued

```

321 2      do forever;
322 3          call rq$wait$interrupt(int$534$level,@ex$val);
323 3          output(IO$534$base+8)=0CH;
324 3          c$level=input(IO$534$base+8) and 07H;
325 3          call rq$send$units(ret$data.int$sema(c$level),1,@ex$val);
326 3          output(IO$534$base+8)=EOI;
327 3      end; /* of do forever */
328 2      end; /* of procedure */

329 1      end interrupt$534$module;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 00B5H      181D
CONSTANT AREA SIZE  = 0000H      0D
VARIABLE AREA SIZE  = 0005H      5D
MAXIMUM STACK SIZE  = 0026H      38D
541 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

Module 8

ISIS-II PL/M-86 X167 COMPILATION OF MODULE IO534TASKMODULE
 OBJECT MODULE PLACED IN :F1:iotask.OBJ
 COMPILER INVOKED BY: plm86 :F1:iotask.plm PRINT(:F5:IOTASK.LST)
 DEBUG COMPACT OPTIMIZE(2) ROM DATE(4/25/80)

```

1      io$534$task$module:
      do;

/*****

      IO$534$TASK: TASK.

      This task receives IORS segments from the queueSio
      procedure and performs the necessary input or
      output operations on the iSBC 534 board.

*****/

      $include(:f4:common.lit)
      = $SAVE NOLIST
      $include(:f1:pointtr.ext)
      = /* external declaration of pointerize procedure */
      = $save nolist
      $include(:f4:nucprm.ext)
      = $SAVE NOLIST
      $include(:f4:nerror.lit)
      =
      = $SAVE NOLIST
      $include(:f1:rettdta.lit)
      = /* literal declaration of ret$data structure for init$534$Sio */
      = $save nolist
      $include(:f1:iors.lit)
      = /* literal declaration for iors */
      = $save nolist

314 1      declare
          begin$io$task$data byte public,
          req$mbox$token external,
          f$detach$device literally '5',
          f$read literally '0',
          f$write literally '1';

315 1      IO$534$task: procedure reentrant public;

316 2      declare
          iors$token,
          iors$p pointer,
          iors based iors$p IO$request$result$segment,
          ex$val word,
          resp$token,
          buff$p pointer,
          buf based buff$p (1) byte,
          i word,
          unit byte,
          ret$data$p pointer,
          ret$data based ret$data$p structure(ret$data$struc),
          c$val word;

317 2      do forever;
318 3          iors$token=receive$message(req$mbox$token,0FFFFH,@resp$token,@ex$val);

/* check for non-existence of mailbox. IF last device has been detached
the mailbox will be deleted In this case, delete thyself */

319 3          if ex$val= E$exist then
320 3              call rq$delete$task(0,@ex$val);
321 3          iors$p=pointerize(iors$token);

```

Module 8, continued

```

322 3      buff$P=iors.buff$P;
323 3      unit=iors.unit;
324 3      iors.actual=0;
325 3      i=0;
326 3      ret$data$P=iors.aux$P;

327 3      if iors.funct = f$detach$device then
328 3          do;
329 4              call rq$sendMessage(
/* mbox token */      resp$T,
/* object token */    iors$T,
/* response token */   0,
/* status ptr */       @ex$Sval);
330 4              call rq$delete$task(0,@ex$Sval);
331 4              end;

332 3      if iors.funct= f$read then
333 3          do while iors.count >0;
334 4              c$val=rq$receive$units(
/* sema */      ret$data.int$sema(2*unit),
/* units */    1,
/* time */     0FFFFH,
/* status*/    @ex$Sval);

335 4              buf(i)=input(ret$data.usart$data$port(unit)) and 07FH;
336 4              i=i+1;
337 4              iors.count=iors.count-1;
338 4              iors.actual=iors.actual+1;
339 4              end;

340 3      else if iors.funct= f$write then
341 3          do while iors.count >0;
342 4              c$val=rq$receive$units(
/* sema */      ret$data.int$sema(2*unit+1),
/* units */    1,
/* time */     0FFFFH,
/* status*/    @ex$Sval);

343 4              output(ret$data.usart$data$port(unit))=buf(i);
344 4              i=i+1;
345 4              iors.count=iors.count-1;
346 4              iors.actual=iors.actual+1;
347 4              end;
348 3      iors.status=E$OK;
349 3      iors.done=TRUE;
350 3      call rq$sendMessage(iors.resp$mbox,iors$T,0,@ex$Sval);
351 3      end; /* of do forever */

352 2      end; /* of procedure */
353 1      end io$534$task$module;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 018DH      397D
CONSTANT AREA SIZE  = 0000H      0D
VARIABLE AREA SIZE  = 0001H      1D
MAXIMUM STACK SIZE  = 0028H      40D
624 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

AP-86

Module 9

ISIS-II PL/M-86 X167 COMPILATION OF MODULE INIT534HW
 OBJECT MODULE PLACED IN :F1:inithw.OBJ
 COMPILER INVOKED BY: plm86 :F1:inithw.plm PRINT(:F5:INITHW.LST)
 DEBUG COMPACT OPTIMIZE(2) ROM DATE(4/25/80)

```

1      init$534$hw:
      do;

/*****

      init$534$hw: PUBLIC PROCEDURE.

      This procedure initializes the iSBC 534 hardware and
      sets up the device dependent fields of the ret$data
      segment which will be used by the queue$io procedures.

*****/

      $include(:f4:common.lit)
=      $SAVE NOLIST
      $include(:f1:ret$dta.lit)
=      /* literal declaration of ret$data structure for init$534$io */
=      $save nolist

12     1      init$534$hw: procedure(ret$data$P) reentrant public;
13     2      declare
           ret$data$P pointer,
           ret$data based ret$data$P structure(ret$data$struc),
           (base,i) byte;

14     2      base=ret$data.io$base;
15     2      output(base+0FH)=0; /* board reset */
16     2      output(base+0DH)=0; /* select data block */
17     2      output(base+8)=16H; /* output ICW1 */
18     2      output(base+9)=0; /* output ICW2 */
19     2      output(base+9)=0; /* output mask word */

      /* attach$device calls will initialize usarts and timers */
      /* set up tables of port addresses for use by queue$io procs */

20     2      ret$data.timer$cmd(0),ret$data.timer$cmd(3)=36H;
21     2      ret$data.timer$cmd(1)=76H;
22     2      ret$data.timer$cmd(2)=0B6H;
23     2      do i=0 to 3;
24     3          ret$data.usart$cmd$port(i)=base+2*i+1;
25     3          ret$data.usart$data$port(i)=base+2*i;
26     3          ret$data.timer$load$port(i)=base+i;
27     3      end;
28     2      ret$data.timer$load$port(3)=base+4;
29     2      ret$data.timer$cmd$port(0),
           ret$data.timer$cmd$port(1),
           ret$data.timer$cmd$port(2)=base+3;
30     2      ret$data.timer$cmd$port(3)=base+7;
31     2      return;
32     2      end;
33     1      end init$534$hw;
  
```

MODULE INFORMATION:

CODE AREA SIZE	= 00E4H	228D
CONSTANT AREA SIZE	= 0000H	0D
VARIABLE AREA SIZE	= 0000H	0D
MAXIMUM STACK SIZE	= 0008H	8D
77 LINES READ		
0 PROGRAM ERROR(S)		

END OF PL/M-86 COMPILATION

Module 8

THIS IS PL/M-86 X167 COMPILATION OF MODULE INIT34SW
 OBJECT MODULE PLACED IN: F:\INIT34SW.08J
 COMPILER INVOKED BY: P1M86 F:\INIT34SW.P1M (F:\INIT34SW.LST)
 DEBBUG COMPACT OPTIMIZE(2) ROM DATE(4/25/88)

INIT34SW: 1
 201

INIT34SW: PUBLIC PROCEDURE.

This procedure initializes the 8280 534 hardware and
 sets up the device dependent fields of the testdata
 segment which will be used by the queueing procedures.

```

    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2
```

FREE SPACE
ROOT JOB
APPLICATION JOB
COMMUNICATIONS JOB
I/O SYSTEM
NUCLEUS
INTERRUPT VECTOR

System Memory Map

```

; ***** NUCLNK.CSD *****
;
; THIS SUBMIT FILE LINKS THE NUCLEUS.
;
;F0:LINK86 &
;F1:NUC86.LIB(NENTRY), &
;F1:NUC86.LIB &
TO :F1:NUCLUS.LNK MAP PRINT(:F1:NUCLUS.MP1) NAME(NUCLEUS)
;
; ***** NUCLOC.CSD *****
;
; THIS SUBMIT FILE LOCATES THE NUCLEUS IN MEMORY.
;
;F0:LOC86 &
;F1:NUCLUS.LNK TO :F1:NUCLUS MAP PRINT(:F1:NUCLUS.MP2) SC(3) &
RESERVE(0 TO 7FFH) SEGSIZE(STACK(0)) &
ORDER(CLASSES(CODE,DATA,STACK,MEMORY)) &
OBJECTCONTROLS(NOLINES,NOCOMMENTS,NOPUBLICS,NOSYMBOLS)

```

Nucleus Link and Locate Commands

```

; ios(date,origin)
;   Sample I/O System .csd file to link and locate an I/O System.
;
; This file links an I/O System with the timer included.
;
; This .csd file assumes the I/O System configuration module is
; iocnfg.a86 (found on the release diskette).
;
; The origin parameter sets the low address of the I/O System;
; all the segments are contiguous in memory.
;
asm86 :fl:iocnfg.a86 date(%0)
link86 &
      :fl:ios.lib(iocinit), &
      :fl:iocnfg.obj, &
      :fl:ios.lib, &
      :fl:rpifc.lib &
to :fl:ios.lnk map print(:fl:ios.mpl)
loc86 :fl:ios.lnk to :fl:ios map sc(3) print(:fl:ios.mp2) &
      oc(noli,nopl,nocm,nosb) &
      order(classes(code,data,stack,memory)) &
      addresses(classes(code(%1))) &
      segsize(stack(0))

```

I/O System Link and Locate Commands

```

;   Submit file to generate located version of file transaction job
;
link86 &
      :fl:ftinit.obj, &
      :fl:listen.obj, &
      :fl:worker.obj, &
      :fl:ptrintr.obj, &
      :fl:rpifc.lib &
to :fl:apex1.lnk map print(:fl:apex1.mpl)
loc86 :fl:apex1.lnk to :fl:apex1 map sc(3) print(:fl:apex1.mp2) &
      oc(noli,nopl,nocm,nosb) &
      order(classes(code,data,stack,memory)) &
      addresses(classes(code(%1))) &
      segsize(stack(0))

```

File Transaction Job; Link and Locate Commands

```

;
;   Submit file to generate located version of communications job
;
link86 &
      :fl:cminit.obj, &
      :fl:comm.lib, &
      :fl:ptrintr.obj, &
      :fl:rpifc.lib &
to :fl:comm.lnk map print(:fl:apex1.mpl)
loc86 :fl:comm.lnk to :fl:comm map sc(3) print(:fl:comm.mp2) &
      oc(noli,nopl,nocm,nosb) &
      order(classes(code,data,stack,memory)) &
      addresses(classes(code(%1))) &
      segsize(stack(0))

```

Communications Job; Link and Locate Commands

077EH	10E4H	PUB	INITDEVICETABLES	077EH	0FBCH	PUB	NAMEDDELETE
077EH	0EB3H	PUB	DECRUSECOUNT	077EH	0E51H	PUB	UNLINKCONN
077EH	0CA8H	PUB	NAMEDCHANGEACCES	077EH	0B5AH	PUB	ATTACHNAMEDFILE
-S							
→ 077EH	073EH	PUB	ATTACHDEVICETASK	077EH	0574H	PUB	ILLEGALFUNCT
077EH	003EH	PUB	RQAIOSINITTASK	077EH	0006H	PUB	COPYRIGHT

SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS
077E0H	1453EH	CD5FH	W	CODE	CODE
14540H	145FFH	00C0H	W	REQ_TABLE	CODE
14600H	146DFH	00E0H	W	IOS_TABLE	CODE
→ 146E0H	14745H	0066H	W	DATA	DATA
14746H	14746H	0000H	W	STACK	STACK
14750H	14750H	0000H	G	??SEG	
→ 14750H	14750H	0000H	W	MEMORY	MEMORY

Locate Map for I/O System

(The "→" indicates entries for job macros and memory map)

1475H	079EH	PUB	SETUP544	1475H	06C5H	PUB	PACKETINPUT
1475H	05B5H	PUB	INDEX	→ 1475H	0572H	PUB	COMMINTTASKENTRY
				-ESS			

SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS
14750H	15BCDH	147DH	W	CODE	CODE
→ 15BD0H	170D2H	1502H	W	DATA	DATA
170D2H	1712EH	004CH	W	STACK	STACK
17130H	17130H	0000H	G	??SEG	
→ 17130H	17130H	0000H	W	MEMORY	MEMORY

Locate Map for Communications Job

17D6H	03B5H	PUB	BEGINLISTENERTASKDATA	1713H	0153H	PUB	POINTERIZE
→ 1713H	0112H	PUB	INITTASKENTRY	1713H	0401H	PUB	WORKERTASK

SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS
17130H	17D59H	0C29H	W	CODE	CODE
17D60H	17E28H	00C8H	W	DATA	DATA
17E30H	17E9AH	006AH	W	STACK	STACK

Locate Map for File Transaction Job

Macro call: SYSTEM (system parameters)			
Number of calls required: exactly one			
CONFIGURATION FILE NAME			
FORMAT:			
	<u>parameter</u>	<u>type</u>	<u>suggested default</u> <u>value</u>
%SYSTEM	(nucleus_entry,	base	80:0
	rod_size,	word	(0) 10
	min_trans_size,	work	(64) 64
	debugger,	see note	(A) N
	default_e_h_provided,	see note	(N) N
	mode)	word	1
NOTES:			
1. Valid entries for the debugger parameter include:			
A Debugger available			
N No debugger available			
2. Valid entries for the default_e_h_provided parameter include:			
Y Yes			
D Debugger			
N No			

Macro call: SAB (for system address blocks)

Number of calls required: one or more

CONFIGURATION FILE NAME: APEX1

FORMAT:

parameter	type	suggested default	value
%SAB	(start_base, end_base, type)	base base see note 1	<u>0</u> <u>1900</u> <u>U</u>

NOTES:

1. The type parameter is reserved for future use. Enter the character U for this parameter.

2. A SAB is declared between start_base:0 and end_base:F, inclusive.

%SAB Macro Worksheet

Macro call: <u>JOB (defines first-level jobs)</u>																																																																		
Number of calls required: <u>one for each first-level job</u>																																																																		
CONFIGURATION FILE NAME: <u>APEX 1</u>																																																																		
<p>FORMAT:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;"></th> <th style="width: 35%; text-align: left;"><u>parameter</u></th> <th style="width: 15%; text-align: left;"><u>suggested type</u></th> <th style="width: 15%; text-align: left;"><u>default</u></th> <th style="width: 20%; text-align: left;"><u>value</u></th> </tr> </thead> <tbody> <tr> <td rowspan="14">%JOB</td> <td>(directory_size,</td> <td>word</td> <td>(0)</td> <td><u>0</u></td> </tr> <tr> <td>pool_min,</td> <td>word</td> <td></td> <td><u>OFFFH</u></td> </tr> <tr> <td>pool_max,</td> <td>word</td> <td>(OFFFH)</td> <td><u>OFFFH</u></td> </tr> <tr> <td>max_objects,</td> <td>word</td> <td></td> <td><u>FFFF</u></td> </tr> <tr> <td>max_tasks,</td> <td>word</td> <td></td> <td><u>FFFF</u></td> </tr> <tr> <td>max_job_priority,</td> <td>byte</td> <td></td> <td><u>129</u></td> </tr> <tr> <td>exception_handler_entry,</td> <td>addr</td> <td>(0:0)</td> <td><u>0:0</u></td> </tr> <tr> <td>exception_handler_mode,</td> <td>byte</td> <td>(1)</td> <td><u>1</u></td> </tr> <tr> <td>job_flags,</td> <td>word</td> <td>(0)</td> <td><u>0</u></td> </tr> <tr> <td>init_task_priority,</td> <td>byte</td> <td></td> <td><u>1713:112</u></td> </tr> <tr> <td>data_segment_base,</td> <td>base</td> <td>(0)</td> <td><u>17D6</u></td> </tr> <tr> <td>stack_pointer,</td> <td>addr</td> <td>(0:0)</td> <td><u>0:0</u></td> </tr> <tr> <td>stack_size,</td> <td>word</td> <td>(512)</td> <td><u>512</u></td> </tr> <tr> <td>task_flags)</td> <td>word</td> <td>(0)</td> <td><u>0</u></td> </tr> </tbody> </table>						<u>parameter</u>	<u>suggested type</u>	<u>default</u>	<u>value</u>	%JOB	(directory_size,	word	(0)	<u>0</u>	pool_min,	word		<u>OFFFH</u>	pool_max,	word	(OFFFH)	<u>OFFFH</u>	max_objects,	word		<u>FFFF</u>	max_tasks,	word		<u>FFFF</u>	max_job_priority,	byte		<u>129</u>	exception_handler_entry,	addr	(0:0)	<u>0:0</u>	exception_handler_mode,	byte	(1)	<u>1</u>	job_flags,	word	(0)	<u>0</u>	init_task_priority,	byte		<u>1713:112</u>	data_segment_base,	base	(0)	<u>17D6</u>	stack_pointer,	addr	(0:0)	<u>0:0</u>	stack_size,	word	(512)	<u>512</u>	task_flags)	word	(0)	<u>0</u>
	<u>parameter</u>	<u>suggested type</u>	<u>default</u>	<u>value</u>																																																														
%JOB	(directory_size,	word	(0)	<u>0</u>																																																														
	pool_min,	word		<u>OFFFH</u>																																																														
	pool_max,	word	(OFFFH)	<u>OFFFH</u>																																																														
	max_objects,	word		<u>FFFF</u>																																																														
	max_tasks,	word		<u>FFFF</u>																																																														
	max_job_priority,	byte		<u>129</u>																																																														
	exception_handler_entry,	addr	(0:0)	<u>0:0</u>																																																														
	exception_handler_mode,	byte	(1)	<u>1</u>																																																														
	job_flags,	word	(0)	<u>0</u>																																																														
	init_task_priority,	byte		<u>1713:112</u>																																																														
	data_segment_base,	base	(0)	<u>17D6</u>																																																														
	stack_pointer,	addr	(0:0)	<u>0:0</u>																																																														
	stack_size,	word	(512)	<u>512</u>																																																														
	task_flags)	word	(0)	<u>0</u>																																																														
<p>NOTE:</p> <p>1. addr is specified as base:offset</p>																																																																		

%JOB Macro Worksheet

```
%sab(0,1900,U)
%job(0,300h,0FFFF,0ffff,0ffff,0,0:0,0,0,128,77e:3e,146e,0:0,512,0)
%job(0,1FFH,0FFFF,0FFFF,0FFFF,128,0:0,0,0,131,1475:572,15bd,0:0,400,0)
%job(0,300H,0FFFF,0FFFF,0FFFF,128,0:0,1,0,130,1713:112,17d6,0:0,400H,0)
%system(80,10,64,N,N,1)
```

Configuration File Apex 1.CNF

```
;
;***** CTABLE.CSD *****
;
; SUBMIT :Fx:CTABLE( fsys, fin, fout, config_file, date )
;
; This submit file assembles the CTABLE module, where:
;   fsys      = the system disk containing ASM86
;   fin       = the source/input disk (F1 is assumed)
;   fout      = the object/listing/output disk
;   config_file = the path-name of the configuration file
;   date      = the date
;
; copy %3 to :f1:config.cnf b
;
;%0:asm86 :%1:ctable.a86 pr(:%2:ctable.lst) oj(:%2:ctable.obj) date(%4) &
xref debug ep
```

Submit File to Generate Configuration Table

```
;
;***** CLNKRJ.CSD *****
;
; SUBMIT :Fx:CLNKRJ( fsys, fin, fout )
;
; This submit file links the Root-Job, where:
;   fsys      = the system disk containing LINK86
;   fin       = the source/input disk
;   fout      = the object/listing/output disk
;
;%0:link86 :%1:croot.lib(root),&
;          :%2:ctable.obj,&
;          :%1:croot.lib &
to :%2:rootjb.lnk map pr(:%2:rootjb.mpl)
```

Submit File to Link the Root Job


```

;***** CLOC RJ.CSD -----
;
; SUBMIT :Fx:CLOC RJ( fsys, fin, fout )
;
; This submit file locates the Root-Job, where:
;   fsys = the system disk containing LOC86
;   fin  = source/input disk
;   fout = object/listing/output disk
;
;-- NOTE: BE SURE TO REPLACE THE "?????" BELOW WITH THE APPROPRIATE
;-- ADDRESS THE ROOT-JOB IS TO BE LOCATED AT!!
;
;%0:loc86 :%2:rootjb.lnk to :%2:rootjb &
map pr(%2:rootjb.mp2) sc(3) &
name(ROOT_JOB) oc(nocm,noli,nopl,nosb) &
segsize(stack(200h)) &
order(classes(data,stack,memory,code)) &
addresses(classes(data(12C00H)))

```

Submit File to Locate Root Job

marking, represents a binary 1 and the absence, or space, of any circuit (an open circuit) represents a binary 0. Both full and half duplex configurations are commonly used.

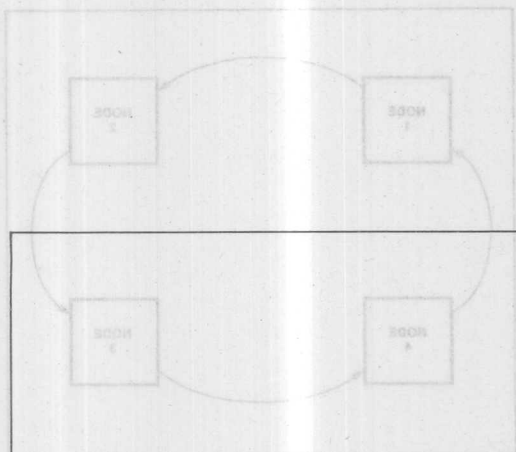


Figure 1. Serial Loop Configuration

The incorporation of loop networks into a system implementation creates its advantages and disadvantages. The strongest argument for using a loop is its inherent noise immunity, which can be gained by using the 20-millisecond current to convey the data. The use of high compliance voltage also allows the transmission of data for large distances at low to moderate transmission rates.

Solid state circuit common to current technology are somewhat less reliable than older mechanical units when the stations are connected onto the loop. This is the result of the requirement to reproduce the information at each node for transmission to the next node. If any one node fails, all other nodes downstream in the communication will be incapable of communicating. However, many good designs incorporate

over long distances with good noise immunity. The loop represents the essence of the communication network. It is a distributed system where all machines connected in the loop are connected in series. Any data sent by one machine is received by all machines connected in the loop. This arrangement is shown in Figure 1. This arrangement is shown in Figure 1. This arrangement is shown in Figure 1.

System designers are seeking more and more difficult application needs with solutions which are microprocessor in a distributed environment. The distribution of intelligence results in many system benefits, including a simplification of the design and a resulting decrease in the development time of the system solution. An additional feature is the minimization of the installation costs of the system resulting from reduced wiring and from resource sharing. However, to effectively create distributed solutions, a reliable communication scheme must be used which links the various parts of the solution.

December 1980

The purpose of this application note is to demonstrate various ways which Intel 180C ports can be configured to implement distributed solutions. Several approaches are offered which apply standard operating modes of both the hardware and of the software. While certainly not the only solution, a distributed processor communications protocol is developed which will support many typical applications.

Distributed systems generally fall into one of two categories. The first consists of those systems in which the processors are located in such a manner as to allow communication over a common system data bus, such as Intel's MULTIBUS™ system bus. The second category involves those systems in which the processors are also geographically distributed. This class of system generally uses a form of communications to allow each processor to communicate with other processors in the system.

The emphasis of this application note is on the second category. Before we discuss the various distributed application solutions, we first discuss the characteristics of application solutions. This note includes a discussion of network design.

Common Network Designs

Serial networks may be classed into three basic categories. These are the LOOP, the MULTIBUS, and the STAR. Each has applicability to certain 180C ports and has distinct advantages and disadvantages in application. After discussing the characteristics of each network, an application scenario illustrates their use.

LOOP NETWORKS The loop represents the essence of the communication network. It is a distributed system where all machines connected in the loop are connected in series. Any data sent by one machine is received by all machines connected in the loop. This arrangement is shown in Figure 1. This arrangement is shown in Figure 1. This arrangement is shown in Figure 1.

Using Intel Single Board Computers for Serial Distributed Processing Links

Peter L. Andersen
OEM Microcomputer Systems
Applications Engineering

INTRODUCTION

System designers are satisfying more and more difficult application needs with solutions which use microprocessors in a distributed environment. This distribution of intelligence results in many system benefits, including a simplification of the design and a resulting decrease in the development time of the system solution. An additional feature is the minimization of the installation costs of the system resulting from reduced wiring and from resource sharing. However, to effectively create distributed solutions, a reliable communication scheme must be used which links the various parts of the solution together.

The purpose of this application note is to demonstrate the ease with which Intel iSBC™ boards can be configured to implement distributed solutions. Several approaches are offered which apply standard operating modes of both the hardware and of the software. While certainly not the only solution, a distributed processor communications protocol is developed which will support many typical applications.

Distributed systems generally fall into one of two categories. The first consists of those systems in which the processors are located in such a manner as to allow communications over a common system data bus, such as Intel's MULTIBUS™ system bus. The second category involves systems in which the processors are also geographically distributed. This class of system generally uses a form of serial communications to allow each processor to communicate with other processors in the system.

The emphasis of this application note is on serial multiprocessing links. Before proceeding with a development of application solutions which involve serial communications, this note includes a short discussion of common network designs.

Common Network Designs

Serial networks may be classed into three general categories. These are the LOOP, the MULTIDROP, and the STAR. Each has applicability to certain iSBC products and has distinct advantages and disadvantages in an application. After discussing the characteristics of each network, an application scenario illustrates their uses.

LOOP NETWORKS

The loop represents the most common of the communication schemes. It is derived from the older teletype communication networks in which several printers were connected in series. Any data generated on one machine causes all machines concerned in the network to echo the data. This arrangement is shown in Figure 1. Loop communications normally use a 20-milliamp signal to convey the data. The presence of this current, known as

marking, represents a binary 1 and the absence, or spacing, of any circuit (an open circuit) represents a binary 0. Both full and half duplex configurations are commonly used.

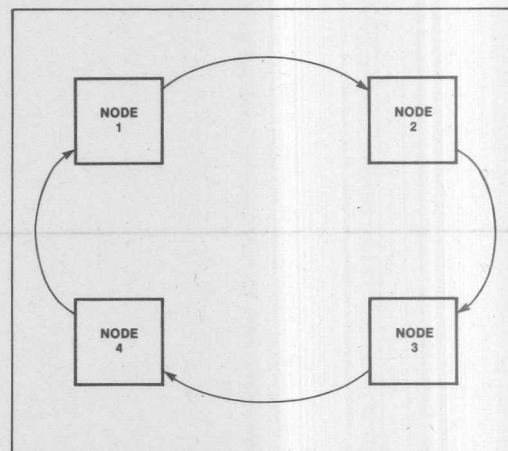


Figure 1. Serial Loop Configuration

The incorporation of loop networks into a system implementation creates both advantages and disadvantages. The strongest argument for using a loop is the inherent noise immunity which can be gained by using the 20-milliamp current to convey the data. The use of a high compliance voltage also allows the transmission of data for large distances at low to moderate transmission rates.

Solid state circuits common to current technology are somewhat less reliable than older mechanical units when many stations are configured onto the loop. This is the result of the requirement to reproduce the information at each node for transmission to the next node. If any node fails, all other nodes downstream in the communication network will not be capable of communicating. Even with this constraint, many good designs incorporate the loop network concepts.

MULTIDROP NETWORKS

Multidrop networks are fast becoming the accepted network for multiprocessor communications. This type of network is shown in Figure 2. As can be seen, this arrangement is similar to that of the loop. The advantage is that all stations are capable of receiving data simultaneously. Multidrop networks are voltage driven since to pass current through a node creates a loop situation. The electrical interface for multidrop networks consists of tri-state drivers and high impedance receivers. The RS422 electrical interface is common for this type of network since it provides high speed data transmission over long distances with good noise immunity.

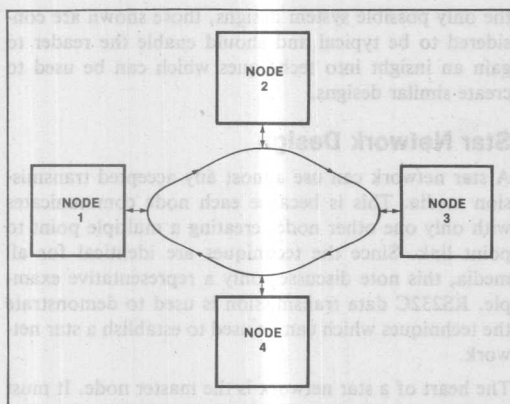


Figure 2. Multidrop Loop Network

The ability to implement networks with a minimum amount of hardware and the potential expansion capabilities provide the multidrop network with a significant advantage in many applications. Since each node in the network can be added by simply adding another tee network, expansion is performed without the need for modification of the communication network.

Intel's iSBX 351™ Serial MULTIMODULE™ Board is an ideal vehicle for the implementation of multidrop networks. Its use is detailed in subsequent paragraphs of this application note.

STAR NETWORKS

Star networks are implemented where either data throughput or hardware limitations prohibit the use of other types of communication arrangements. Figure 3 illustrates a typical star network implementation. Note that a star network is really an example of multiple point to point communications. System throughput is improved since communication can occur with all slave nodes simultaneously without the need to have each node monitor traffic which is not intended for it.

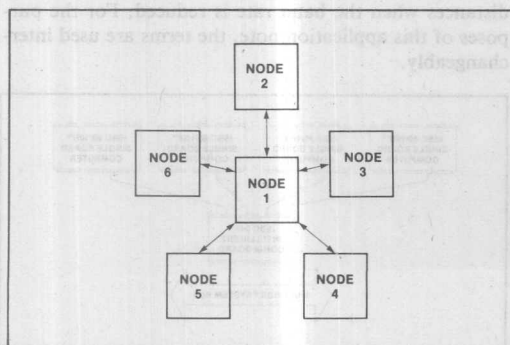


Figure 3. Star Configuration

Intel's iSBC 544™ Intelligent Communications Controller and the iSBC 534™ Four-Channel Communications Board are ideal choices to design star communications networks. The outer points of the star are implemented using any single board computer which has on-board serial communications. An example of this type of network is included in this application note.

Sample Application

An application example is used in this application note to illustrate the techniques which are required to implement various types of serial communication networks. Both hardware and software aspects of the design are described in some detail.

The application discussed in this note involves the creation of an alarm and security system for a multiple building complex. This complex is shown in Figure 4. The solution generated allows monitoring of three existing buildings with provision for the addition of a fourth at a later date. The figure shows that each building is to have a local annunciator panel for reporting activity in those areas served by the building sensors. In addition, a main security station provides monitoring of all buildings in the complex.

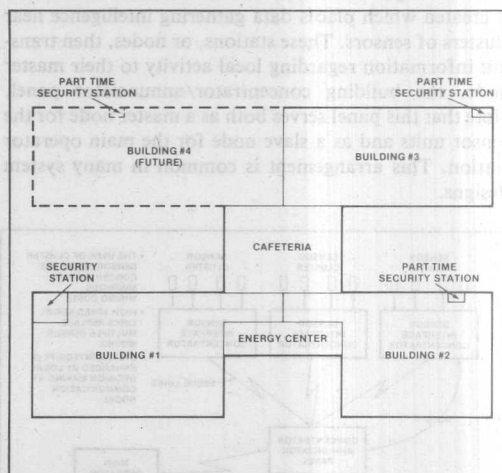


Figure 4. Application Building Complex

Applications such as this are optimized by distributing the system intelligence throughout the complex. This serves two needs; that of minimizing the wiring costs, and of providing an improvement in system reliability. When a system is distributed in this way, a means must be devised which allows communication between the various elements of the network. Both hardware network design and software communication protocol are required before the system can be considered operational.

In order to illustrate various network designs, the example is broken into a distributed system as shown in Figure 5. This is an example of a multidrop communication network. The system design does not require that building nodes communicate with each other; all communications are between building annunciator nodes and the main operator station. This illustrates the concept of a master and a slave. This will be further discussed in a later portion of this application note.

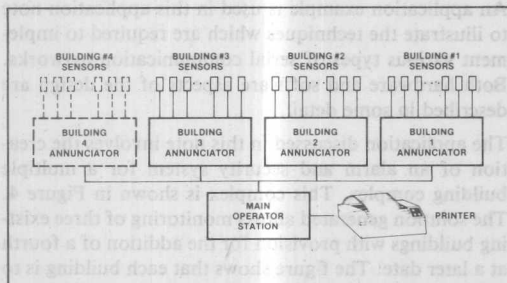


Figure 5. A Multidrop Application

The building annunciator can be further distributed as shown in Figure 6. Here, a star communication network is created which places data gathering intelligence near clusters of sensors. These stations, or nodes, then transmit information regarding local activity to their master node, the building concentrator/annunciator panel. Note that this panel serves both as a master node for the sensor units and as a slave node for the main operator station. This arrangement is common in many system designs.

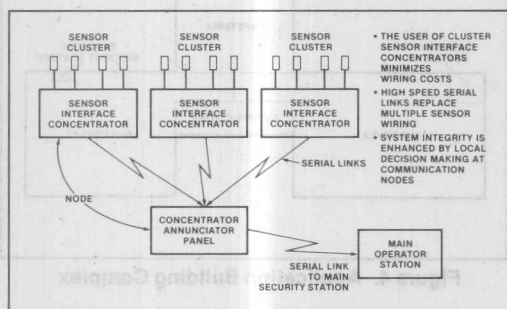


Figure 6. A Star Application

HARDWARE IMPLEMENTATION

The implementation of communication networks using iSBC boards for both star and multidrop techniques is detailed in the following sections. In each case, a master/slave relationship is assumed. For purposes of illustration, actual data from the application example is used in performing the sample calculations. While not

the only possible system designs, those shown are considered to be typical and should enable the reader to gain an insight into techniques which can be used to create similar designs.

Star Network Design

A star network can use almost any accepted transmission media. This is because each node communicates with only one other node, creating a multiple point to point link. Since the techniques are identical for all media, this note discusses only a representative example. RS232C data transmission is used to demonstrate the techniques which can be used to establish a star network.

The heart of a star network is the master node. It must provide a means of independent communication with each of its slave nodes. Costs can be minimized if multiple communication drivers are placed onto the same board. The example chosen requires that four slave nodes be controlled by the master. The iSBC 544 Intelligent Communications Controller provides this function. It provides a software selectable baud rate and the ability to offload the host processor of communication activities. If additional slave nodes are required, the system can be expanded by adding iSBC 544 controller boards.

An ideal choice for a slave node is the iSBC 80/10B™ Single Board Computer. This computer provides good processing capabilities with low cost. It includes both an RS232C and 20-milliamp current loop communications interface. For many small applications it makes an ideal choice for a complete single board solution.

Figure 7 shows the implementation for a four-node plus master star communication network for the security application. Both the iSBC 544 communications board and the iSBC 80/10B Single Board Computer require jumpering and component insertion to allow operation in either the RS232C mode or in the EIA mode. EIA operation is electrically identical to RS232C but has specifications which allow transmissions at significant distances when the baud rate is reduced. For the purposes of this application note, the terms are used interchangeably.

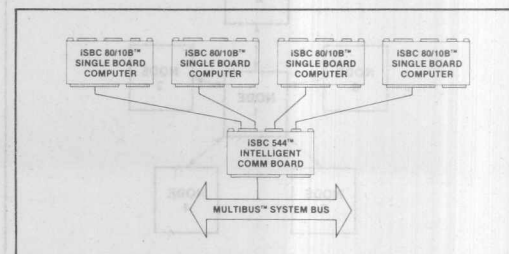


Figure 7. Star Implementation

In the case of the iSBC 544 board, a header plug can be inserted into an on-board connector to enable the correct mode of operation. The wiring for this header is shown in Figure 8. Note that the ready to send and the clear to send signals are jumpered to allow the correct operation of the USART without the control signal of a modem. Also note that the transmit and receive signals are reversed through the header.

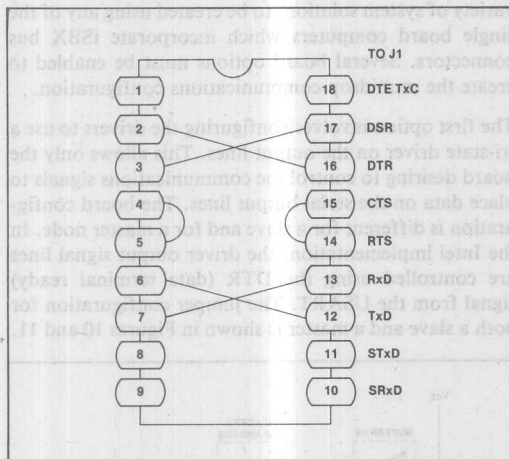


Figure 8. Master to Master Wiring Header.

The use of an RS232C signal for long distance communication necessitates the use of a low baud rate for reliable data transmissions. This is shown in Figure 9.

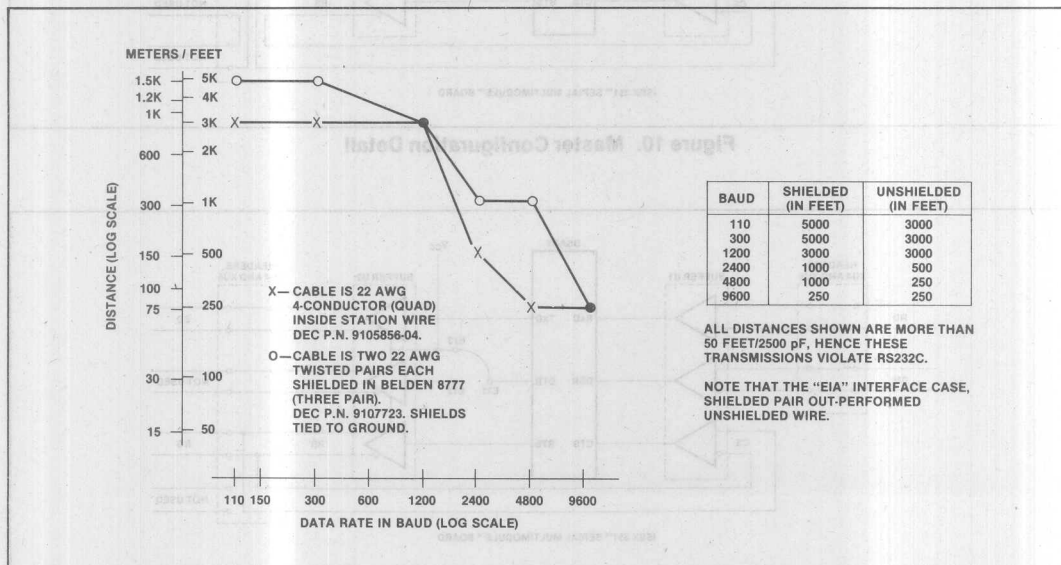


Figure 9. RS232 Baud Rate vs Distance

The security system application design allows for minimum communication traffic so a baud rate of 1200 is used for the star networks. The placement of intelligence at the sensor cluster minimizes the amount of data which must be transmitted, providing adequate response time at this data rate.

The hardware design phase includes the selection of the communication scheme to be used between the host processor and the slave node. Fortunately, this task is simplified by the use of built-in master/slave protocol functions contained on the iSBC 544 communications board. These functions involve three hardware features. The first is the incorporation of dual-ported memory onto the board. This memory provides a media for the storage of data and commands for communication between the slave and the host processor. The second feature generates a flag interrupt each time the host board writes into the first location of the dual-ported memory. The interrupt is cleared when the memory location is read by the slave processor. Finally, the execution of the 8085A-2 SOD instruction generates a MULTIBUS interrupt to inform the host board of the need to service the slave. This interrupt can be vectored to the host interrupt matrix to activate a service routine in support of the slave board.

A detailed explanation of the use of these features is found in another application note, AP-53, *Using the iSBC 544™ Intelligent Communications Controller*. The reader should refer to this document for details of the board and its use.

Initial network Design

The complexity of a multinode system is minimized through the use of a multidrop network. Certain limitations are placed on the acceptable hardware transmission media in multidrop applications. A scheme must be used which allows each of the nodes to alternatively gain control of the network in order to communicate its message.

Both half and full duplex configurations are allowable in a multidrop network. While it is simpler from a hardware standpoint, a half duplex connection requires more stringent communications protocol as this system allows communications in only one direction at a time. For all practical purposes, no priority for masters or slaves is provided. All nodes may listen to whomever is using the line at the time. The software must be written to allow only one node to communicate at a given instant. An example of a half duplex network is the Ethernet.

More straightforward software can be written for full duplex communication configurations. In this instance,

an assumption is made that only one master is configured into the system and always drives the output lines. Any number of slaves can be placed to drive the input lines, communicating only when queried by the master node. This is the scheme used for the application example which is discussed in this application note.

Intel's iSBX 351 Serial MULTIMODULE Board lends itself well to a multidrop application. It allows a wide variety of system solutions to be created using any of the single board computers which incorporate iSBX bus connectors. Several board options must be enabled to create the multidrop communications configuration.

The first option involves configuring the drivers to use a tri-state driver on the output lines. This allows only the board desiring to control the communications signals to place data on the serial output lines. The board configuration is different for a slave and for a master node. In the Intel implementation, the driver output signal lines are controlled using the DTR (data terminal ready) signal from the USART. The jumper configuration for both a slave and a master is shown in Figures 10 and 11.

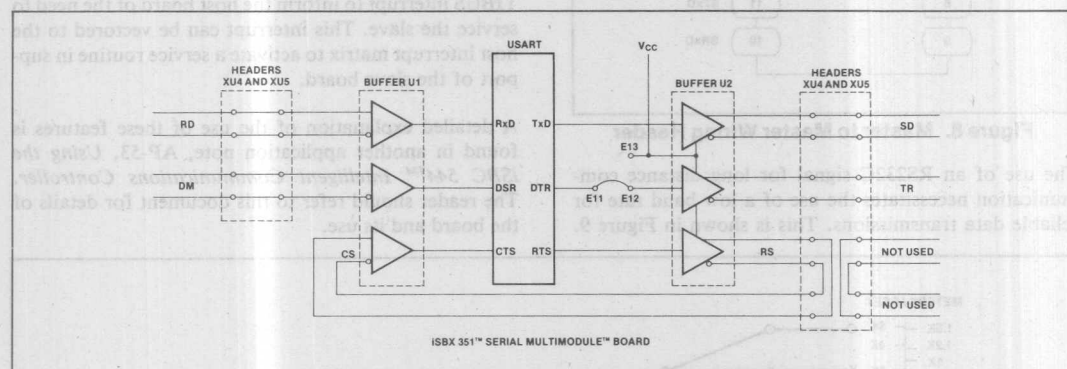


Figure 10. Master Configuration Detail

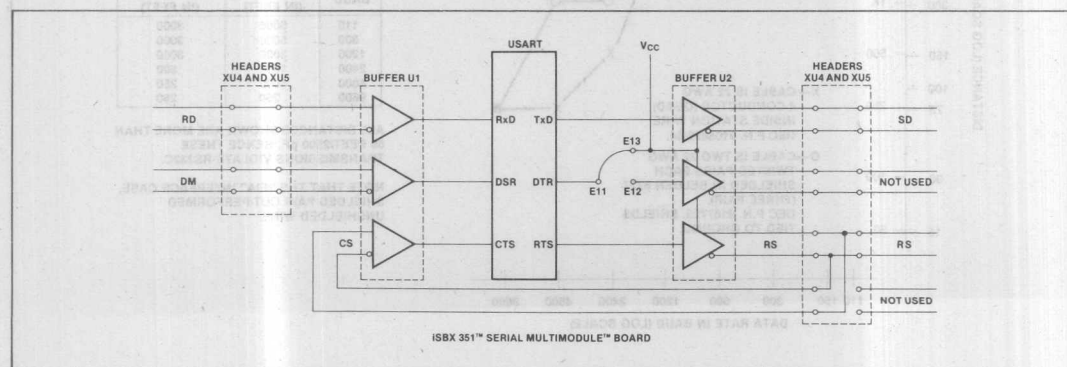


Figure 11. Slave Configuration Detail

Figure 12 illustrates the physical wiring data paths which are used to create a full duplex multidrop network having a master and four slave stations. Each board must be configured using a header block to provide the correct signals. This is done as shown in Figures 10 and 11.

Finally, bias resistors are chosen to terminate the serial communication data paths. A discussion of the formulas for determining the correct values for the two resistors is provided in Appendix A of the *iSBX 351™ Serial MULTIMODULE Board Hardware Reference Manual* (Order Number 9803190). If the equations are solved for the components used on the RS422 section of the board, the equations for calculating the correct values become:

$$Rb1 = 4500 / (18.6 - 1.6n)$$

where n is the number of slave nodes

AND

$$Rb2 = 2500 / (18.6 - 1.6n)$$

where n is the number of slave nodes.

The drivers used on the iSBX 351 communications MULTIMODULE board limit the number of slave nodes to 11 for any one network. If greater numbers of nodes are required, a driver component substitution may be required.

For the example application in this note, four slave nodes are used, so the calculated resistor values become 370 ohms for Rb1 and 205 ohms for Rb2.

Figure 13 illustrates the acceptable baud rates for various communications cable lengths using the RS422 in-

terface. The iSBC 351 board is designed to support data transmission rates of up to 19.2 kilobaud. Thus, the system designed using these boards is seen to permit a total multidrop cable length of up to 4000 feet (1.219 km). The sample application discussed in this note incorporated a total communications link length of 1600 feet, well within the performance limits.

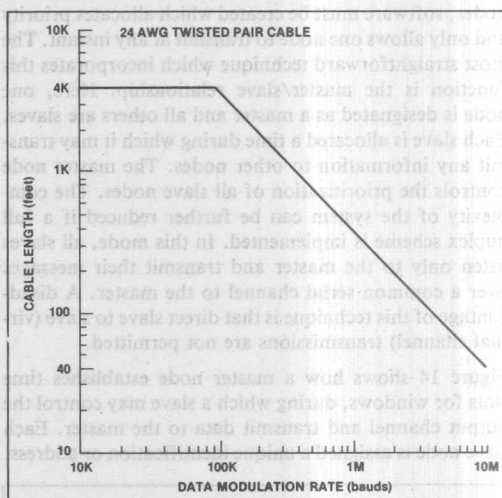


Figure 13. RS422 Data Rate vs Distance

SOFTWARE IMPLEMENTATION

Once the hardware has been defined, the design process moves into a software phase. Here, protocols are defined which provide both data transmission and handshaking between nodes. Handshaking is defined as background communication which maintains synchronization and integrity of the communications link.

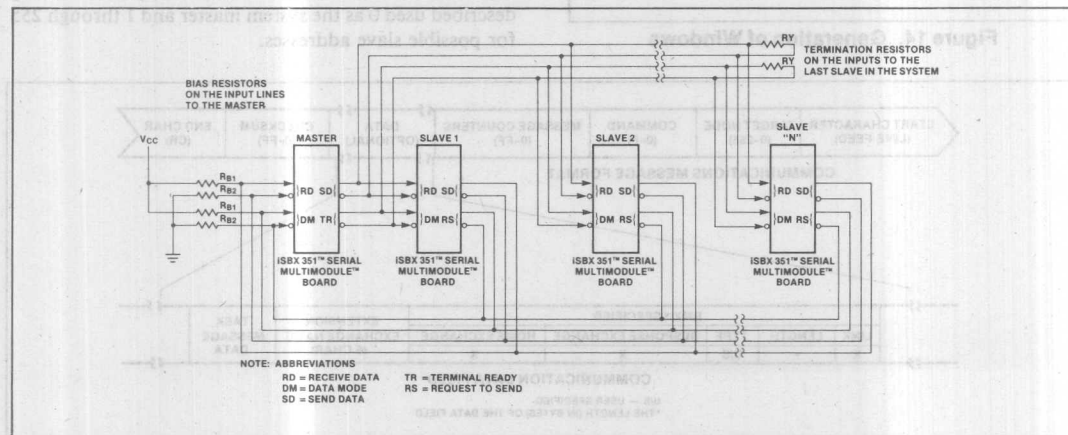


Figure 12. Master/Slave Physical Wiring

Again, standard features of both iSBC board products and of iRMX™ software products simplify the incorporation of communication protocols into the system solution.

Master/Slave Relationships

In order to maintain an orderly flow of data between nodes, software must be created which allocates priority and only allows one node to transmit at any instant. The most straightforward technique which incorporates this function is the master/slave relationship. Here, one node is designated as a master and all others are slaves. Each slave is allocated a time during which it may transmit any information to other nodes. The master node controls the prioritization of all slave nodes. The complexity of the system can be further reduced if a full duplex scheme is implemented. In this mode, all slaves listen only to the master and transmit their messages over a common serial channel to the master. A disadvantage of this technique is that direct slave to slave (virtual channel) transmissions are not permitted.

Figure 14 shows how a master node establishes time slots for windows, during which a slave may control the output channel and transmit data to the master. Each slave node is assigned a unique identification or address.

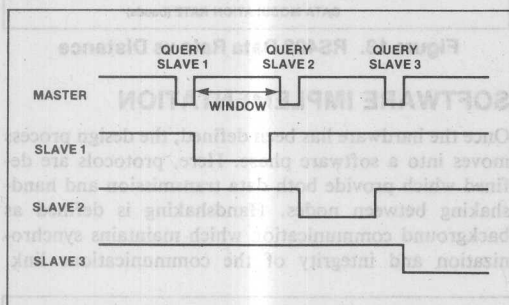


Figure 14. Generation of Windows

Each data packet contains an address as an integral part. When a slave recognizes its address in the packet from the master, it knows that it has fixed time window in which to return its own data packet to the master.

Data Packet Formats

Communications between each slave and its master involve the transmission of data packets. Each packet contains both handshaking information (such as checksums and message counts) and the information which is to pass between nodes. The handshaking portions of the message are used to maintain the integrity of transmissions in the face of such external stimuli as electrical noise.

Debugging and system maintenance is minimized when a means is provided which allows easy viewing and interpretation of the communication messages. The scheme used in this application note uses a fully ASCII compatible communications format. This allows a CRT terminal to tap onto the link and to monitor all communication messages.

Each message packet begins and ends with a unique character. This character cannot exist with a message block. Using the ASCII formats, all messages use the numeric representation of 0 to 9 and the alpha characters from A to Z. A line feed is used to begin a message packet and a carriage return is used to terminate the message. The assignment of these characters assures an easily interpreted message packet.

The layout of the data packet is shown in Figure 15. The justification and description of each field is given in the following paragraphs. Note that much of the message is concerned with maintaining system integrity. The requirement for an address field has already been described. Two characters are used for this field and provide for 256 addresses (0-FF hex). The protocol being described used 0 as the system master and 1 through 255 for possible slave addresses.

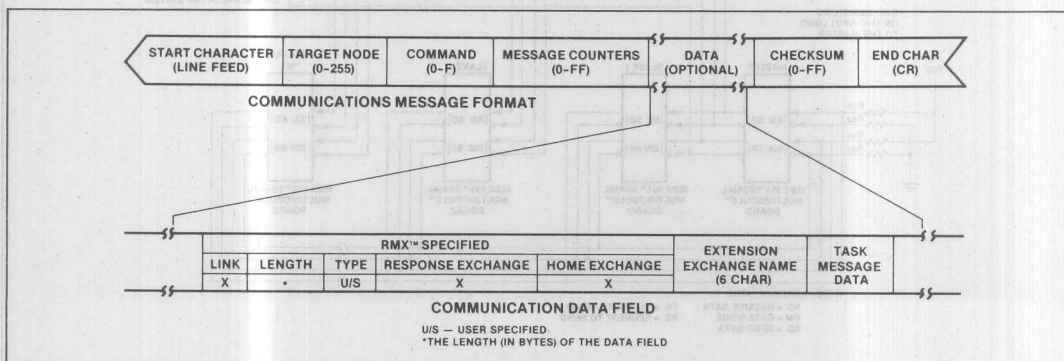


Figure 15. Communications Packet Format

The next field indicates the type of message being sent and the action which is required by the receiver. Analysis of system communication requirements indicate that five message types are sufficient. The use of a single byte of the packet for this field allows a total of 16 message types. This leaves 11 for future expansion. The types being used in this application are:

Type 0 — This is a reset request from the master to a slave. It will cause the target slave to generate a software reset. This command is used only when communications cannot be established using other means.

Type 1 — A type 1 command is used to synchronize the master and the slave. It will cause the message counters (see subsequent paragraphs) to reset to zero. This command is issued only by the master node.

Type 2 — The type 2 message is defined to be that which contains information which is to be moved between nodes of the communications network. The exact format of the data is discussed elsewhere. This is the only message which is not considered to be of a strictly handshaking nature.

Type 3 — A type 3 message is used to indicate that a network node is responsive and ready to handle messages. When normal communications have been established and no data messages are being sent, this message is continuously being sent between the master and the slaves.

Type 5 — The final message type is the type 5. It is used by the slave to respond to the master's request to synchronize. The receipt of this message indicates that the slave has performed all necessary operations with its message counters and is ready to receive messages.

The next field is used to describe the number of messages which contain data and have been received or sent by each node. The first character is used to indicate the number of messages which have been received from the master node to that slave. The second character indicates the number of messages which have been sent from the master to the node. Internal software protocol enables the communications package to use these fields to determine if a data message has been received correctly by the target node. If a response message does not contain the correct counts, the message can be retransmitted. Since only one character is used for each count, the numbers are allowed to recycle each 15 messages.

The data field contains information which is to be sent to the receiving node. Its exact content is discussed later in the application note during the discussion of the iRMX 80 message extensions.

A checksum is included to guarantee the integrity of the transmitted message. This field contains the 8-bit modulo 256 sum of all transmitted ASCII characters, beginning with the line feed and continuing through the last character of the data field. It is used to compare a calculated checksum of received characters with that transmitted by the sending node. If these two numbers are in agreement, the message is considered to be valid and can be used by the receiving node's application software.

The final field is the end of message character. A carriage return is used and provides a unique indicator of the end of a message packet.

The implementation of these communication concepts into a functioning software package is illustrated later during the discussion of the layering of the multiprocessing communications package.

Multitasking Message Concepts

The design of systems which involve distributed processors strongly suggests that a multitasking environment is also present. The ability to segregate an application into tasks allows a considerable simplification of the design and implementation process. In reality, few tasks represent completely isolated functions. Some degree of communications is required between the tasks. This may range from a simple synchronization of tasks to a data transfer. Real-time executives simplify these processes for the system designer. Intel's iRMX 80 nucleus is an excellent example of such an executive. It provides low overhead, small physical size, and operates on almost all 8-bit Intel single board computers. The product has been thoroughly covered in several application notes and it is therefore not necessary to cover it again in this note. However, certain features are useful for developing a distributed processor extension of the basic nucleus.

An iRMX 80 message is analogous to a letter which is mailed to someone. It is sent via a mailroom or post office (called an exchange) and is then picked up by the receiving party. Each part of the iRMX implementation can be thought of in this manner. When a serial communications link is used between the processors, the analogy translates to that of sending a mailgram. The following short discussion deals with the structure and mechanism for sending messages between tasks which reside on different single board computers. For clarity, the mail analogy is used.

There are several distinct parts of a message. Each has a function and a format within the message structure. Before a message can be generated, a medium must be procured to contain that message. In terms of a large office, this medium is paper; in the multitasking system, it is a block of RAM. In order to maintain a continuing supply of memory, the iRMX executive provides the

user with what is known as a free space manager. The purpose of the free space manager is to recycle memory blocks so that they can be again used for generation of additional messages. This service is called each time the sending task requires to generate a message. When the communication programs have successfully transmitted the message to the target node, the memory block is again returned to the free space memory pool. As the target node receives the message it must obtain a block of memory from its free space manager to store the information until it can be processed by the target task. The target task returns the block to the pool when it has taken the appropriate actions.

As with any type of message, the individual to whom the message is to be delivered must be provided. Because the iRMX 80 nucleus is designed to be used by multiple tasks on the same processor, the target address mechanism is not included in the message header itself; instead, the target is specified in the call to the iRMX procedure as a parameter list component. When the target exchange address is not known, which happens when multiprocessor applications are created, it is necessary to create an extension to the executive which allows the assignment of a target exchange name to each message. Fortunately, this is relatively easy to create.

iRMX 80™ Named Exchange Extension

The iRMX 80 nucleus uses a small block of RAM memory to store data regarding the characteristics and status of each exchange. This storage area normally occupies 10 bytes of memory. Its layout is shown in Figure 16. Except to note that a unique identification field does not exist which sets each exchange descriptor apart, it is beyond the scope of this application note to dwell on the meanings of the fields. However, a method must be created which allows the extension of the field to include an identification which is unique to that descriptor.

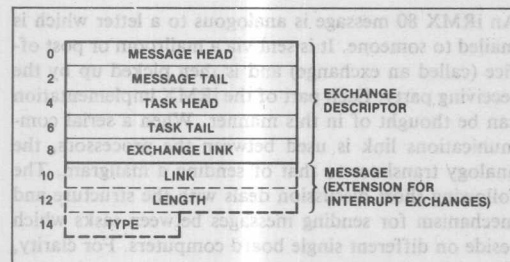


Figure 16. Exchange Descriptor

If one is not to rewrite the nucleus, the format of the exchange descriptor must not be changed and the identification field should be added as additional bytes of the memory block. Two features of the iRMX 80 nucleus make the direct implementation of this concept impossible. First, a special exchange descriptor already exists

within the standard real-time executive. This is the interrupt exchange which adds 5 bytes to the nucleus for use as an interrupt message. Second, if an additional extension were to be added, all exchanges would have to be created as interrupt exchanges so that common software could be used. While this would not in itself be prohibitive, the interactive configurator (ICU 80) does not allow addition of fields to either the standard or to the interrupt exchange descriptor. Another method is required to add the extension.

Fortunately, another method exists to create an exchange. The nucleus operation, RQCXCH, creates an exchange at an address which is specified by a parameter of the call instruction. If user software is created to build the named exchanges, a name can be prefixed to each desired name exchange.

In the standard iRMX 80 nucleus six characters are allowed to name each task. A logical extension of this concept provides a six-character name for each named exchange. The exchange descriptor is thus extended by inserting 6 bytes before the basic format. In reality, this is not sufficient because named exchanges are to be created dynamically. The memory blocks representing the set of named exchanges to not necessarily constitute a contiguous block of RAM. This leads to the creation of an additional word of memory to carry a pointer to the next named exchange field. A value of zero in this field indicates that no additional named exchanges exist. If a non-zero value is placed into the field, it is a pointer to the next named exchange. Figure 17 illustrates the structure of the named exchange fields.

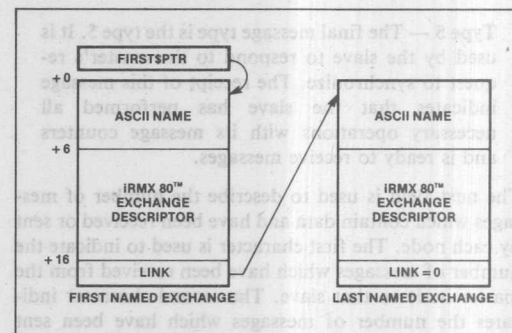


Figure 17. Named Exchange Descriptors

The creation of the software to support the named exchange extension is rather straightforward. Two functions are required; one which creates the named exchanges, and a second which will return the address of an exchange which matches a given name. A complete listing of the software generated to perform these functions is shown in Appendix A. The services of the free space manager are used to allocate the memory segments which are used for the exchange descriptors. A

quick examination of the program techniques provides some insight into the operations involved in creating extensions to the standard nucleus.

Examination of the listing at line 55 shows that a procedure is used to allow user code to determine the absolute address of the iRMX exchange descriptor field from the named exchange lists. This address can in turn be used with either the RQWAIT or the RQSEND instruction in the user's code. If no corresponding name is found in the table, a value of zero is returned by the procedure. An address parameter is passed in the user call to the FIND\$EXCH subroutine which points to the 6 bytes containing the ASCII name of the referenced exchange.

An internal address value, identified as FIRST\$PTR, contains the address of the first named exchange descriptor. A simple DO loop sequence is used to locate a matching name in the table lists. Either a zero or the location of an actual 10-byte exchange descriptor within the extended memory block is returned.

A task is used to provide the system capability of building named exchanges. A task, rather than a procedure, is included because a task allows the initialization of system pointers such as FIRST\$PTR. The named exchange building task, CREATE\$COM, waits at its input exchange, COM\$CREATE\$EXCH until a request for creation of a named exchange is received from a user task. The listing for this task is found in Appendix A, beginning on line 78.

When a message is received, the task obtains a 20-byte block of memory from the free space manager. It then moves the ASCII name from the requesting message into the appropriate fields and uses the iRMX primitive call, RQCXCH, to create an exchange descriptor within the memory block. The pointer field of the last named exchange is updated to point to the new memory field and the pointer field of the newly created named exchange is set to zero.

Finally, the address of the actual exchange descriptor is returned to the requesting program as an updated parameter of the request message.

The creation of named exchanges greatly enhances the capabilities of systems designed around the iRMX 80 nucleus. This extension allows the generation of distributed systems in which tasks may communicate with each other regardless of their geographical locations so long as some type of link exists.

Generation of Multiprocessing Serial Communications Link

The creation of software for a serial communications link provides the capability of allowing true multitasking, multiprocessor capabilities. The need for two types of communications packages, a slave and a master, indicates that two separate tasks are required. A compre-

hensive examination of the communication requirements indicates that the system can be generated in three layers. Figure 18 shows the layer relationships. The first is defined as the protocol and consists of that code which is required to implement the algorithms for either the slave or the master network nodes. The second level is known as the link level and contains that code which is used to support common operations such as message generation and data queue handling. The third provides a specialized interface to the physical hardware which is being used to generate the communications link. This level, called the physical level, is unique to each configuration.

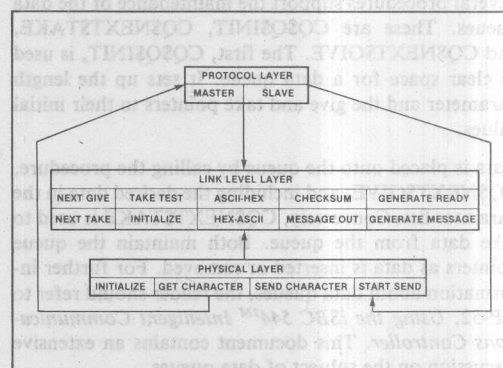


Figure 18.

PROTOCOL LEVEL COMMUNICATIONS PACKAGE

Two protocol level communications packages are included in this application note. One implements the master protocol and is reproduced in Appendix B. The second, the slave protocol, is found in Appendix C.

No attempt is made to detail the operations which are involved in the task generation. The code is commented and is readily followed by the reader who has an understanding of PL/M programming techniques. Some facts relating to the implementation of the packages as iRMX 80 tasks are in order.

The master communications package is designed to use a USART device which is configured to provide interrupts at level 7 each time a character is received. A physical level interrupt handler supports the receipt of each individual character, and, when an end of message character (carriage return) is received, places an interrupt message into the interrupt exchange, RQL7EX. At this point (Appendix B, line 244), the master protocol handler can begin processing the received message. Because the task is associated with interrupt level 7, a task priority in the range between 113 and 128 must be assigned to the task. The example used in this application note uses a priority of 113.

In a similar manner, the slave communications task uses interrupt level 6 to receive messages (Appendix C, line 130). Its priority must lie in the range between 97 and 112.

LINK LEVEL COMMUNICATIONS PACKAGE

The link level communications package provides a common set of procedures which can be used to support both the protocol and the physical layers. Listings of these support programs are found in Appendix D. The programs which make up the package are defined in the following discussion.

Several procedures support the maintenance of the data queues. These are CQ\$Q\$INIT, CQ\$NEXT\$TAKE, and CQ\$NEXT\$GIVE. The first, CQ\$Q\$INIT, is used to clear space for a data queue. It sets up the length parameter and the give and take pointers to their initial values.

Data is placed onto the queue by calling the procedure, CQ\$NEXT\$GIVE, and including the desired data in the parameter list. Conversely, CQ\$NEXT\$TAKE is used to take data from the queue. Both maintain the queue pointers as data is inserted or removed. For further information about data queues, the reader should refer to AP-52, *Using the iSBC 544™ Intelligent Communications Controller*. This document contains an extensive discussion on the subject of data queues.

Two procedures deal with the transformation of data between printable ASCII and the internal binary format. CQ\$ASC\$HEX transfers data from the data queue into a working buffer. In the process, it converts the format from ASCII into a hex representation usable by the target task. Likewise, CQ\$HEX\$ASC is used to transform data in a user buffer into a transmittable format. In both cases, appropriate start and stop characters are added or deleted as necessary to create the correct format.

One procedure deals with computing the checksum of a message which is in ASCII format. This procedure, CQ\$CHECKSUM, returns a zero if the checksum agrees with that in the message. If an error is indicated, a value of -1 (OFF hex) is returned.

Finally, two procedures support the generation of message packets. One, CQ\$GEN\$RDY, is used to build a "ready" message by creating the appropriate data into the fields. The second, CQ\$GEN\$MSG, generates a data message containing an iRMX 80 message to be sent to an exchange on another processor board.

PHYSICAL LEVEL COMMUNICATIONS PACKAGE

The physical level communications package provides the customization for the unique configuration of host

processor and USART. Four procedures must be included with this level. These are:

- **CQ\$INIT** — This public procedure contains all operations necessary to initialize the timers, counters and USARTs associated with the communications code. In addition, this procedure defines the interrupt service routines for the USART and enables the corresponding interrupt levels.
- **CQ\$START\$MSG** — A public procedure which places the USART into a mode in which the transmitter is enabled. The execution of this procedure should result in an interrupt being generated by the USART transmitter ready line.
- **CQ\$MIVT** — This is an interrupt service routine associated with the USART receiver ready line. It is entered each time that a character has been received by the communication node. In the case of a slave node, this procedure is known as CQ\$IVT. The name is unimportant because the location of the routine is passed to the iRMX 80 nucleus at initialization by the CQ\$INIT program. When a carriage return (end of message) is encountered, a message is passed to the protocol level task by passing a message to the interrupt exchange, RQL6EX, using the iRMX primitive, RQISND.
- **SEND\$CHAR** — Like the CQ\$MIT routine, this is an interrupt handler procedure. It is entered each time the USART signals that it is ready to transmit a character. A new character is obtained from the data queue and it is transmitted to the receiving node. If the character is the end of message, flags are set and the USART transmitter is disabled.

The listing of a sample physical driver is provided in Appendix E. This driver illustrates the use of the iSBX 351 Serial MULTIMODULE Board placed into a socket of an iSBC 80/24 Single Board Computer.

The example from the application implements a multi-drop slave node. The enabling of the tri-state drivers is accomplished in line 138 by sending the USART a command of 025H. When the message has been sent, the driver is again placed into a high impedance mode by sending a command of 026H (line 121). These commands control the driver by toggling the DTR or Data Terminal Ready lines of the 8251A USART device.

APPLICATION EXAMPLE

The alarm and security system previously discussed provides a perfect environment in which to use the concepts developed in this application note.

To verify the functionality of the communications package, the transmissions between a master and a slave were monitored using a CRT terminal. The terminal was

The ability to break applications into small tasks, either functionally or geographically, aids the system designer by allowing his concentration on each task. A message transfer capability allows these tasks to again be tied together to form a complete solution to the application. Where the tasks reside on the same single board computer, the standard iRMX nucleus provides this ability. When different host boards are used, extensions to the nucleus allow the same functions to be performed. Both multiple processors on the same MULTIBUS chassis (refer to AP-88 and AP-112) and in different chassis using the serial links described in this application note can be supported with extensions to the nucleus.

Many concepts are used to create serial communication networks. Intel's single board computer products simplify the design process. Among these products, several stand out in this application. For example:

- **iRMX 80™ Executive** — This outstanding product simplifies the design of multitasking systems and provides a foundation for the implementation of extensions to create many varied configurations. Such things as task synchronization, interrupt handling, and message transfers allow multitasking. The free space manager allocates RAM and is an integral part of the serial communications link software. The terminal handler and the disk operating system simplify the software design by providing ready to use I/O drives which can be accessed by the user.
- **iSBC 80/10B™ Single Board Computer** — This microcomputer allows a low cost solution to many small to medium applications. The ability to operate the board in an iRMX 80 environment enhances its capabilities by allowing it to multiprocess. Its on-board serial communications link allows it to be used as a slave node in either EIA or current loop communications networks. The iSBX MULTIMODULE connector further enhances its capabilities by allowing customization of I/O to meet a wide variety of user applications.

- **iSBC 544™ Intelligent Communications Board**

— The use of this board allows much of the protocol and physical drivers to be off-loaded from the host processor. The board provides an ideal master communications node for star type networks. By placing the handshaking operations on the communication board, the host is free to perform application-oriented functions with a much higher throughput rate.

- **iSBC 80/24™ Single Board Computer**

— This powerful 8085A-2 based microcomputer board provides the capability to implement most of the system functions without the need to expand to additional memory or I/O expansion MULTIBUS boards. It fully supports the iRMX 80 executive. Its ability to act as a full bus master allows even greater flexibility through the implementation of MULTIBUS multiprocessor solutions. The two iSBX MULTIMODULE expansion connectors provide the user with considerable flexibility in the design of his system. The use of one of these sockets as a carrier for the serial communications MULTIMODULE board makes a multidrop serial communications link practical.

- **iSBX 351™ Serial MULTIMODULE Board**

— The implementation of multidrop communications requires the use of an electrical interface which supports more than one communications node to share the link. The use of the RS422 operating mode of the board easily implements this feature. A well-defined hardware protocol also assists in the implementation of a serial multidrop communications link. Up to 11 slave nodes can be designed into a system using this powerful board. In addition, the iSBX 351 board can function in either the RS232C or the EIA mode for linking into terminals or for creating a point to point network.

The assistance of Judy McMillan in the implementation and testing of the security system and its communication links is greatly appreciated.

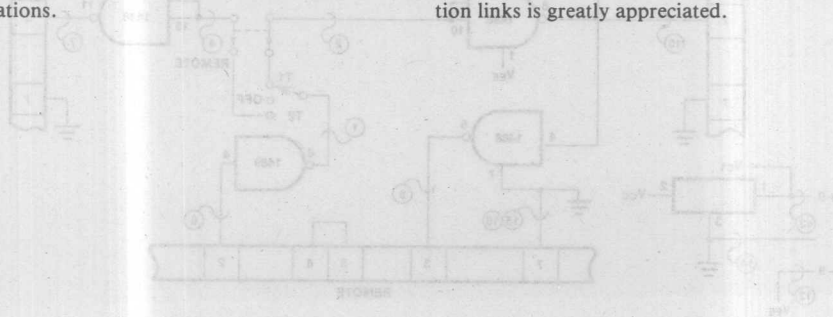


Figure 18. Communications Test Schematic

```

title RMX\88 EXTENSION TO NAME EXCHANGES, VERSION 1.3
createcomsexch;
do;
*****
CREATEEXCHANGE creates a list of exchanges associating
the name of an exchange with its location, for the purpose
of other tasks wishing to find the location of an exchange.
FINDSEXCHANGE, when given the name of an exchange, points
down the list and, when it finds a match, returns the
address to the requesting task.
*****
201list

```

```

/* declare exchanges used by the task */
declare COM$CREATESEXCH exchange$description public;
declare CREATESEXCH exchange$description public;
/* declare pointers and messages used by the task */

```

APPENDIX A	2-189
APPENDIX B	2-193
APPENDIX C	2-205
APPENDIX D	2-215
APPENDIX E	2-236

```

declare firstptr address;
declare lastptr address;
declare msgptr address;
declare exchptr address;
declare byte;
declare exch based exchptr structure;
declare lastexch based lastptr structure;
declare response based msgptr structure;
exchange address;
name;
procedure(exptr) address public;
declare addr address;
declare exptr address;
declare (ex based exptr) (6) byte;
/* declare the returning message with the address */

```



```

1      $title('RMX/80 EXTENSION TO NAME EXCHANGES, VERSION 1.3')
      create$com$exch:
      do;
/*****
CREATE$EXCHANGE creates a list of exchanges associating
the name of an exchange with its location, for the purpose
of other tasks wishing to find the location of an exchange.
FIND$EXCHANGE, when given the name of an exchange, polls
down the list and, when it finds a match, returns the
address to the requesting task.
*****/
$no list

      /* declare exchanges used by the task */

26 1      declare COM$CREATE$EXCH exchange$descriptor public;
27 1      declare CREATE$EXCH      exchange$descriptor public;

      /* declare pointers and messages used by the task */

28 1      declare first$ptr address;
29 1      declare last$ptr  address;
30 1      declare msg$ptr   address;
31 1      declare exch$ptr  address;
32 1      declare n         byte;

33 1      declare request based msg$ptr structure(
          msg$hdr,
          exch$name(6) byte);

34 1      declare fs$req structure(
          msg$hdr,
          msg$length address);

35 1      declare exch based exch$ptr structure(
          name(6)      byte,
          exchange(10) byte,
          link         address);

36 1      declare last$exch based last$ptr structure(
          name(6)      byte,
          exchange(10) byte,
          link         address);

37 1      declare response based msg$ptr structure (
          msg$hdr,
          exchange address );

38 1      NAMEX:
      Procedure(exptr) address public;
39 2          declare addr address;
40 2          declare exptr address;
41 2          declare (ex based exptr) (6) byte;

      /* declare the returning message with the address */

```

```

42  2      declare ret$msg based addr  structure(msg$hdr,
        exch$adr  address);
        /* declare message to send to create a named exchange */
43  2      declare req structure(msg$hdr,
        name(6) byte);

        /* create an exchange to communicate with
        the comm create exchange */

44  2      declare fsx exchangedescriptor;
45  2      call rqcxch(.fsx);

        /* build and send the request message */
46  2      req.length = 15;
47  2      call move(6, .ex, .req.name);
48  2      req.type = 200;
49  2      req.response$exchange = .fsx;
50  2      call rqsend(.comcreate$exch, .req);
51  2      addr = rqwait(.fsx, 0);
52  2      addr = ret$msg.exch$adr;

        /* return with the address of the exchange */

53  2      return(addr);
54  2      end;

55  1      FIND$EXCH:
        procedure (name$ptr) address public;

        /* *****
        This procedure finds the exchange having a name specified
        by the passed parameter pointer. If a match is found the
        exchange address is returned. If no match is found, a
        zero is returned.
        ***** */

56  2      declare name$ptr address;
57  2      declare match byte;
58  2      declare (name based name$ptr)(6) byte;

        /* test for no exchanges */
59  2      if first$ptr = f
        then return 0;

        /* test for a name match */
61  2      else do;
62  3          exch$ptr = first$ptr;
63  3          do while 1;
64  4              match = 0ffh;
65  4              do n = f to 5;
66  5                  if name(n) <> exch.name(n)

```

```

        then match = 0;
68 5      end;
69 4      if match then return .exch.exchange;
71 4      if exch.link = 0 then return 0;
73 4      exch$ptr = exch.link;
74 4      end;
75 3      end;
76 2      return 0;
77 2      end;

78 1      CREATE$COM:
      procedure public;

      /*****
      This procedure creates exchange extensions when asked to
      do so by a task that wants its exchange made completely
      accessible. It saves the name of the exchange then
      creates an exchange to save its address. It may be updated
      at any time.
      *****/

      /* initialize exchanges */
79 2      call rqcxch(.com$create$exch);
80 2      call rqcxch(.create$exch);

      /* initialize pointers */
81 2      last$ptr = 0;
82 2      first$ptr = 0;

83 2      do while 1;

          /* get a request for creation of an exchange */
84 3      msg$ptr = rwait(.com$create$exch, 0);

          /* get some memory for the exchange */
85 3      fs$req.length = 11;
86 3      fs$req.type = 5;
87 3      fs$req.response$exchange = .create$exch;
88 3      fs$req.msg$length = 20;
89 3      call rsend(.rqfsax, .fs$req);
90 3      exch$ptr = rwait(.create$exch, 0);
91 3      exch$ptr = fs$req.msg$length;

          /* store name in the exchange structure */
92 3      call move( 6, .request.exch$name(0),
          .exch.name(0));

          /* create an exchange */
93 3      call rqcxch(.exch.exchange(0));

          /* update the link field */

94 3      if last$ptr > 0
96 4      then do;
          last$exch.link = exch$ptr;

```

```

97 4 last$ptr = exch$ptr;
98 4 exch.link = 0;
99 4 end;
100 3 else do;
101 4 exch.link = 0;
102 4 first$ptr = exch$ptr;
103 4 last$ptr = exch$ptr;
104 4 end;

/* return the exchange pointer */
105 3 response.exchange = .exch.exchange;
106 3 call rqsnd(request.response$exchange, msg$ptr);

107 3 end;
108 2 end;
109 1 end;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 01D3H    467D
VARIABLE AREA SIZE  = 0048H    72D
MAXIMUM STACK SIZE  = 0004H    4D
244 LINES READ
C PROGRAM ERROR(S)

```



```

1  $title('Master Communications Protocol Driver, Version 8.1')
    MASTERSDRIVER: do;
    /*
    The task contained in this listing supports the master
    communications protocol for establishing network commun-
    ications.
  
```

The task must be configured using the ICU/PC as follows:

```

    TASK NAME: MASTER
    TASK ENTRY POINT: MLINK
    TASK PRIORITY: 113
    TASK STACK LENGTH: 100
  
```

```

    EXCHANGE: ROL7EX
    SCOPE: EXTERNAL
    INTERRUPT: YES
    EXCHANGE: TIMEX
    SCOPE: PUBLIC
    INTERRUPT: NO
  
```

```

    LINK: :Fn:CQMSTR.OBJ
  
```

```

    LINK: :Fn:CQCOM.LIB
    LINK: :Fn:CQCSLV.LIB
  
```

Tasks desiring to send a message to a node should send a message to the exchange CQMIEX(node).

Certain include files are used by the task.

```

    */
    $include (:ff:exch.elt)
2  1  =  DECLARE EXCHANGESDESCRIPTOR LITERALLY 'STRUCTURE (
    =  MSG$HEAD ADDRESS,
    =  MSG$TAIL ADDRESS,
    =  TASK$HEAD ADDRESS,
    =  TASK$TAIL ADDRESS,
    =  NEXT$EXCH ADDRESS)';
    $include (:ff:ied.elt)
3  1  =  DECLARE INT$EXCHANGESDESCRIPTOR LITERALLY 'STRUCTURE (
    =  MESSAGE$HEAD ADDRESS,
    =  MESSAGE$TAIL ADDRESS,
    =  TASK$HEAD ADDRESS,
    =  TASK$TAIL ADDRESS,
    =  EXCHANGE$LINK ADDRESS,
    =  LINK ADDRESS,
    =  LENGTH ADDRESS,
    =  TYPE BYTE)';
    $include (:ff:msq.elt)
4  1  =  DECLARE MSG$HDR LITERALLY '
  
```

```

= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
5 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
= MSG$HDR,
= REMAINDER(1) BYTE)';
= $include (:f0:common.ext)
6 1 = DECLARE TRUE LITERALLY '0FFH';
7 1 = DECLARE FALSE LITERALLY '00H';
8 1 = DECLARE BOOLEAN LITERALLY 'BYTE';
9 1 = DECLARE FOREVER LITERALLY 'WHILE 1';
= $include (:f0:synch.ext)
10 1 = RQSEND:
= PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;
11 2 = DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
=
12 2 = END RQSEND;
=
13 1 = RQWAIT:
= PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
14 2 = DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
=
15 2 = END RQWAIT;
=
16 1 = RQACPT:
= PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
17 2 = DECLARE EXCHANGE$POINTER ADDRESS;
=
18 2 = END RQACPT;
=
19 1 = RQISND:
= PROCEDURE (IED$PTR) EXTERNAL;
20 2 = DECLARE IED$PTR ADDRESS;
=
21 2 = END RQISND;
= $include (:f0:intrpt.ext)
22 1 = RQENDI:
= PROCEDURE EXTERNAL;
=
23 2 = END RQENDI;
=
24 1 = RQELVL:
= PROCEDURE (LEVEL) EXTERNAL;
25 2 = DECLARE LEVEL BYTE;
=
26 2 = END RQELVL;
=
27 1 = RQDLVL:
= PROCEDURE (LEVEL) EXTERNAL;
28 2 = DECLARE LEVEL BYTE;
=
29 2 = END RQDLVL;

```

```

30 1 = RQSETV:
    = PROCEDURE (PROC,LEVEL) EXTERNAL;
31 2 = DECLARE PROC ADDRESS;
32 2 = DECLARE LEVEL BYTE;
    =
33 2 = END RQSETV;
    =
34 1 = RQSETP:
    = PROCEDURE (PROC,LEVEL) EXTERNAL;
35 2 = DECLARE PROC ADDRESS;
36 2 = DECLARE LEVEL BYTE;
    =
37 2 = END RQSETP;
    $include (:ff:fsmgr.ext)
38 1 = DECLARE RQFSAX EXCHANGESDESCRIPTOR EXTERNAL;
39 1 = DECLARE RQFSRX EXCHANGESDESCRIPTOR EXTERNAL;
40 1 = DECLARE RQFREE EXCHANGESDESCRIPTOR EXTERNAL;
    $include (:fl:masmsg.elt)
41 1 = declare msg$format literally 'structure (
    = trgt byte,
    = cmdnd byte,
    = seq$la byte,
    = seq$na byte,
    = text(250) byte )';
    =
42 1 = declare queue$format literally 'structure (
    = end$ptr byte,
    = give$ptr byte,
    = take$ptr byte,
    = data$byte(254) byte )';
    =
43 1 = declare par$format literally 'structure (
    = na byte,
    = la byte,
    = run byte,
    = stop byte,
    = que$pt byte )';
    $include (:ff:objman.ext)
44 1 = RQCTSK:
    = PROCEDURE (STATIC$PTR) EXTERNAL;
45 2 = DECLARE STATIC$PTR ADDRESS;
    =
46 2 = END RQCTSK;
    =
47 1 = RQCXCH:
    = PROCEDURE (EXCHANGES$PTR) EXTERNAL;
48 2 = DECLARE EXCHANGES$PTR ADDRESS;
    =
49 2 = END RQCXCH;
    $include (:fl:comcom.ext)
50 1 = CQ$NEXT$GIVE:
    = Procedure (q$ptr,give$byte) byte external;
51 2 = Declare q$ptr address;
52 2 = Declare give$byte byte;

```

```

53 2 =      end cq$next$give;
54 1 =      CQ$TAKE$TEST:
      =      Procedure (q$ptr) byte external;
55 2 =      Declare q$ptr address;
56 2 =      end cq$take$test;
57 1 =      CQ$NEXT$TAKE:
      =      Procedure (q$ptr) byte external;
58 2 =      Declare q$ptr address;
59 2 =      end cq$next$take;
60 1 =      CQ$Q$INIT:
      =      Procedure (q$ptr,q$size) external;
61 2 =      Declare q$ptr address;
62 2 =      Declare q$size byte;
63 2 =      end cq$q$init;
64 1 =      CQ$ASC$HEX:
      =      Procedure (q$ptr,msg$ptr) external;
65 2 =      Declare (q$ptr,msg$ptr) address;
66 2 =      end cq$asc$hex;
67 1 =      CQ$CHECKSUM:
      =      Procedure (q$ptr) byte external;
68 2 =      Declare q$ptr address;
69 2 =      end cq$checksum;
70 1 =      CQ$HEX$ASC:
      =      Procedure (q$ptr,hex$byte) external;
71 2 =      Declare q$ptr address;
72 2 =      Declare hex$byte byte;
73 2 =      end cq$hex$asc;
74 1 =      CQ$MSG$OUT:
      =      Procedure (msg$size,q$ptr,par$ptr,msg$ptr) external;
75 2 =      Declare msg$size byte;
76 2 =      Declare (q$ptr,par$ptr,msg$ptr) address;
77 2 =      end cq$msg$out;
78 1 =      CQ$GEN$RDY:
      =      Procedure (trget,msg$ptr,q$ptr,par$ptr) external;
79 2 =      Declare trget byte;
80 2 =      Declare (msg$ptr,q$ptr,par$ptr) address;
81 2 =      end cq$gen$rdy;
82 1 =      CQ$GEN$MSC:
      =      Procedure (msg$pointer,trget,msg$ptr,q$ptr,
      par$ptr,save$flg) external;
83 2 =      Declare (trget,save$flg) byte;
84 2 =      Declare (msg$pointer,msg$ptr,q$ptr,par$ptr) address;
85 2 =      end cq$gen$msg;

86 1      USRINT:
      Procedure external;
87 2      end usrint;
88 1      FIND$EXCH:
      Procedure (name$ptr) address external;
89 2      declare name$ptr address;
90 2      end find$exch;
91 1      NAMEX:
      Procedure (exptr) address external;
92 2      declare exptr address;
93 2      end namex;

```



```

94 1      return$ram:
          procedure(pointer) external;
95 2      declare pointer address;
96 2      end return$ram;

/*
    Certain literals are used to define the network's
    physical characteristics. These are:
*/
97 1      Declare N$NODE literally '0'; /* number of nodes */
98 1      Declare NODE$ARRAY literally '5';

/*
    A structure provides the data queues for the
    transmission of data to each node. It is defined
    and is available as a public.
*/
99 1      Declare CQMSTQ queue$format public at (811ch);

/*
    A single data queue is used to support the input of
    data from a node since only one slave is given a
    window at a time:
*/
100 1     Declare CQMSIQ queue$format public at (801bh);

/*
    Each node has an associated set of flags which
    indicate the operational mode of that node. The
    function of each bit is defined as:
        bit 0 - request synchronization (synch$request)
        bit 1 - request initialization (init$request)
        bit 2 - repeat last message (repeat$request)
        bit 3 -
        bit 4 -
        bit 5 -
        bit 6 -
        bit 7 - error flag (error$flag)
*/
101 1     Declare CQCMTDF(node$array) byte public;
102 1     Declare SYNCH$REQUEST literally 'C1H';
103 1     Declare INITS$REQUEST literally 'C2H';
104 1     Declare REPEAT$REQUEST literally 'C4H';
105 1     Declare ERROR$FLAG literally 'CCH';

/*
    A counter is used to indicate the number of attempts
    to establish communications with a node. When it
    reaches a maximum value, a synch will be sent to the
    node. when the maximum value of synchs are sent to a
    node with no effect, a reset will be sent to the node.
*/

```

```

106 1      Declare ERROR$COUNT(node$array) byte;
107 1      Declare MAX$COUNT literally '5';
108 1      Declare SYNCH$COUNT(node$array) byte;

      /* a pointer is used to store the address of exchanges
      used by incoming messages */

109 1      Declare target$exch address;
      /*
      A set of exchanges is used to hold a message until it
      has been acknowledged by the slave.
      */
110 1      Declare HOLD$EXCH(node$array) exchange$descriptor;
      /*
      A set of exchanges is provided to accept messages which
      are to be sent to any of the nodes.
      */
111 1      Declare CQMEX(node$array) exchange$descriptor public;
      /*
      The task uses various sets of data structures
      */
112 1      Declare req$msg structure (
      msg$hdr,
      msg$length address);
113 1      Declare (m,n,node$cnt,outmode,ram$size) byte;
114 1      Declare msg$ptr address;
115 1      Declare msg msg$format;
116 1      declare start$544 byte public at (8000h);
117 1      Declare data$block based msg$ptr structure (
      msg$hdr);
118 1      Declare CQMPAR(node$array) par$format public
      at (8002h);
119 1      Declare RAM$MSG based msg$ptr structure (
      msg$hdr);
120 1      Declare CQ$ACTIVE$NODE byte public at (8001h);
121 1      Declare fre$mem address;
122 1      Declare fre$msg based fre$mem structure(msg$hdr);
      /*
      The task uses certain exchanges for internal
      communications
      */
123 1      Declare CQMSEX exchange$descriptor;
124 1      Declare RQL7EX int$exchange$descriptor public;
      Select
      /******
      error$test
      This short procedure is used to increment the error
      counters and to signal an initialization or synch command
      when required. It will order a re-transmission of the
      last message each time an error is detected.
      *****/

```

```

125 1      error$test: procedure;
126 2          If (error$count(n):=error$count(n)+1) > max$count
           then do;
128 3              error$count(n)=0;
129 3              if (synch$count(m):=synch$count(m)+1) > max$count
           then do;
131 4                  synch$count(m) = 0;
132 4                  cqcndf(n) = cqcndf(n) or error$flag
           or init$request;
133 4              end;
134 3              else cqcndf(n) =cqcndf(n)
           or error$flag or synch$request;
135 3          end;
136 2          else cqcndf(n) = cqcndf(n) or repeat$request;
137 2      return;
138 2      error$test;
           Seject
139 1 MLINK: Procedure public;

           /* Initialize the system and the task */
140 2      Do n=0 to n$node;
141 3          CQCNDf(n)=synch$request or error$flag;
142 3          CQMPAR(n).run,CQMPAR(n).stop,CQMPAR(n).que$pt=0;
143 3          ERROR$COUNT(n)=0;
144 3          SYNCH$COUNT(n)=0;
145 3      end;

           /* Initialize the node pointer */
146 2      NODE$CNT=0;

           /* Initialize the input exchanges */
147 2      Do N=0 to n$node;
148 3          call RQCXCH(.CQMIEX(n));
149 3          call RQCXCH(.HOLD$EXCH(n));
150 3      end;

151 2      call RQCXCH(.CQSEX);

           /* Initialize the queues */
152 2      call cq$init(.cqmstp, 254);
153 2      call cq$init(.cqmsta, 254);

           /* Create the named exchanges and give ram to fsm*/
154 2      call usrint;

           /* initialize the level 7 interrupt exchange */
155 2      call rqlvl(7);

           /* Begin main task loop */
156 2      Do forever;

           /* Begin support of one nodal channel */
157 3      Do n=1 to n$node;

```

```

158 4      /* reset queue pointers */
      camstq.give$ptr, camstq.take$ptr = 0;

159 4      /* Compute nodal mode number */
      If (cacmdf(n) and synch$request)>0
161 4      then outmode = 1;
162 4      else outmode = 0;
      If (cacmdf(n) and init$request)>0
164 4      then outmode = 2;
      cq$active$node = n;

      /* if comm failure, kill all messages */
165 4      if (cacmdf(n) and error$flag) > 0
      then do;
167 5          do while (msg$ptr:=rqacpt(.comex(n))) > 0;
168 6              call return$ram(msg$ptr);
169 6          end;

170 5      do while (msg$ptr:=rqacpt(.hold$exch(n)))
      > 0;
171 6          call return$ram(msg$ptr);
172 6      end;
173 5      end;

      /* Operate on nodal mode */
174 4      Do case outmode;

      /* case 0, routine communications */
175 5      Do;
176 6      If (cacmdf(n) and repeat$request)>0
      then do;

      /* support of retransmission request */
178 7      cacmdf(n)=cacmdf(n)
      and not repeat$request;
179 7      msg$ptr=rqacpt(.hold$exch(n));
180 7      if msg$ptr>0

      /* retransmit old message */
      then do;
182 8          if campar(n).na = 0
184 8          then campar(n).na = 15;
          else campar(n).na=campar(n).na
          - 1;
185 8          call catgen$msg(msg$ptr,n,.msg,
          .camstq,.campar(n),0);
186 8          call rqsnd(.hold$exch(n),
          msg$ptr);
187 8          if campar(n).na = 15
189 8          then campar(n).na = 0;
          else campar(n).na
          =campar(n).na+1;
190 8      end;

```



```

191 7      /* no old message, send ready */
192 8      else do;
193 8          msg.trgt=n;
194 8          msg.cmd=3;
195 8          call cq$msg$out(0,.cqmsta,
196 7          .cqmpar(n),.msg);
197 6      end;
198 7      /* end of retransmission */
199 7      /* support of next message */
200 6      else do;
201 8          /* clear hold exchange */
202 8          msg$ptr=rqacpt(.hold$exch(n));
203 8          if msg$ptr>0
204 8          then do;
205 8              /* free space return size */
206 7              ramsize=data$block.length;
207 7              if (ramsize mod 4) > 0
208 7              then data$block.length
209 7              =data$block.length
210 7              +(4-(ramsize mod 4));
211 8              call ROSEND(.rqfsrx,msg$ptr);
212 8          end;
213 7          /* test for a message output */
214 7          msg$ptr=rqacpt(.cqmiex(n));
215 7          /* when a message exists */
216 7          if msg$ptr>0
217 7          then do;
218 8              if (target$exch:
219 8              =find$exch(msg$ptr +0))> 0
220 8              then call
221 8              rgsend(target$exch,msg$ptr);
222 8          else do;
223 8              call cq$gen$msg(msg$ptr,n,.msg,
224 8              .cqmsta,.cqmpar(n),0);
225 8              call rgsend(.hold$exch(n),
226 8              msg$ptr);
227 8          /* increase the na flag */
228 8          if cqmpar(n).na = 15
229 8          then cqmpar(n).na = 0;
230 8          else cqmpar(n).na
231 8          = cqmpar(n).na + 1;
232 8          end;
233 7          end;
234 7          /* when no message exists */
235 7          else do;
236 8              msg.trgt=n;
237 8              msg.cmd=3;

```

```

222 8      call cq$msg$out(0,.comsta,
223 8      end;
224 7      end; /* end of routine message */
          /* start the message transmission */
225 6      start$544 = n;
226 6      end; /* end of case 0 */
          /* case 1, synch request */
227 5      do;
228 6      cqmpar(n).na,cqmpar(n).la = 0;
229 6      msg.trgt=n;
230 6      msg.cmd=1;
231 6      call cq$msg$out
          (0,.comsta,.cqmpar(n).msg);
232 6      start$544 = n;
233 6      cqcmdf(n)=cqcmdf(n)
          and not synch$request;
234 6      end;
          /* case 2, initialize request */
235 5      do;
236 6      cqcmdf(n)=(cqcmdf(n) and
          not init$request) or synch$request;
237 6      cqmpar(n).na,cqmpar(n).la = 0;
238 6      msg.trgt=n;
239 6      msg.cmd=0;
240 6      call cq$msg$out
          (0,.comsta,.cqmpar(n).msg);
241 6      start$544 = n;
242 6      end;
243 5      end; /* end of do case blocks */
          /* wait for a response from the slave */
244 4      msg$ptr = rql7ex,25);
245 4      start$544 = 0;
          /* test for a message or a timeout */
246 4      if ram$msg.type=3
          /* support of no valid response */
          then
          /* test for max number of errors to node */
247 4      call error$test;
          /* support of good response return */
248 4      else do;
          /* test for good checksum */

```

```

249 5 if cqschecksum(.cqmseq)=0
    then do;

251 6 /* convert message to hex format */
    call cq$asc$hex(.cqmseq,.msg);

252 6 /* test for data message receipt */
    if msg.cmdnd = 2

    /* support receipt of data message */
    then do;

    /* get ram from free space manager */
254 7 req$msg.length=11;
255 7 req$msg.type = 5;
256 7 req$msg.response$exchng=.cq$ms$ex;
257 7 req$msg.msg$length=msg.text(2);
258 7 call rsend(.rqf$ax,.req$msg);
259 7 msg$ptr=rowait(.cq$ms$ex,0);
260 7 msg$ptr=req$msg.msg$length;

    /* move message into memory */
261 7 call move(msg.text(2),.msg.text(0)
    ,msg$ptr);

    /* if target is another node.. */
262 7 if msg.trgt > 0
    then call rsend(.cqmi$ex(msg.trgt)
    ,msg$ptr);

    /* if target is master board */
264 7 if (target$exch:= find$exch(msg$ptr +
    9))
    then
265 7 call rsend(target$exch, msg$ptr);
266 7 else do;
267 8 ransize = msg.text(2);
268 8 if (ransize mod 4) > 0
    then msg.text(2)=msg.text(2)
    + (4 - (ransize mod 4));
270 8 call rsend(.rqf$rx, msg$ptr);
271 8 end;

    /* increment message counter */
272 7 if ccmpar(n).la = 15
    then ccmpar(n).la = 0;
274 7 else ccmpar(n).la=ccmpar(n).la+1;

275 7 end; else
    /* respond to non-sequence acknowledge */
276 6 if msg.cmdnd=5

```

```

then cqm$par(n).la,cqm$par(n).na = 0;
/* test for a good slave LA response */
278 6 if cqm$par(n).na = msg.seqla + 1
then call error$test;
280 6 else do;
281 7 if msg.seqla <> cqm$par(n).na
then cqm$df(n) = cqm$df(n)
or synchrequest;
283 7 else do;
284 8 error$count(n) = 1;
285 8 cqm$df(n) = cqm$df(n)
and not error$flag;
286 8 end;
287 7 end;end;
288 6 end; /* end of response return */
/* support of bad checksum */
289 5 else call error$test;
290 5 end;
291 4 end; /* end of message arrived options */
292 3 end; /* end of one node */
293 2 end; /* end of task */
294 1 end

```

Several system parameters must be defined in the user code to define the node characteristics. These parameters are defined as:

nodeid - a byte value providing the node address of the communication node.

freemem - an address value which provides a pointer to a block of RAM to be assigned via the free space manager to the communications.

ramsize - an address value which indicates the size of the above block.

```

*
declare cqm$par byte external;

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```



```

1 $title('Slave Communications Package Version 3.2')
COMSLVSMODULE: do;
/*
This is the slave communication main task.
It should be configured into the system
having the following parameters:
Task name- CQSLAV
Task entry point- CQSLAV
Task priority- as required
Task stack length- 100

The LINK and LOCATE commands of the user
must include the following statements:
Link ...
...
:Fn:CQSLAV.OBJ
:Fn:CQS351.OBJ
:Fn:CQCOM.LIB
:Fn:CQSLV.LIP
...

```

Several system parameters must be defined in the user code to define the mode characteristics. These parameters are defined as:

ca\$slad- a byte value providing the nodal address of the communication node.
 fre\$mem- an address value which provides a pointer to a block of RAM to be assigned via the free space manager to the communications.
 ram\$len- an address value which indicates the size of the above block.

```

*/
2 1 declare cqlslad byte external;

$no1ist
$include(:f1:commsg.elt)
48 1 = declare msg$format literally 'structure (
=      trgt      byte,
=      cmnd      byte,
=      seq$la     byte,
=      seq$na     byte,
=      text(250) byte );'
=
49 1 = declare queue$format literally 'structure (
=      end$ptr    byte,
=      give$ptr   byte,
=      take$ptr   byte,

```

```

= data$byte(254) byte );
=
50 1 = declare par$format literally 'structure (
= la byte,
= na byte,
= run byte,
= stop byte,
= quespt byte );
= $include(:ff:objman.ext)
51 1 = RQCTSK:
= PROCEDURE (STATIC$PTR) EXTERNAL;
52 2 = DECLARE STATIC$PTR ADDRESS;
=
53 2 = END RQCTSK;
=
54 1 = RQCXCH:
= PROCEDURE (EXCHANGE$PTR) EXTERNAL;
55 2 = DECLARE EXCHANGE$PTR ADDRESS;
=
56 2 = END RQCXCH;
= $include(:fl:comcom.ext)
57 1 = CQ$NEXT$GIVE:
= Procedure (q$ptr,give$byte) byte external;
58 2 = Declare q$ptr address;
59 2 = Declare give$byte byte;
60 2 = end cq$next$give;
61 1 = CQ$TAKE$TEST:
= Procedure (q$ptr) byte external;
62 2 = Declare c$ptr address;
63 2 = end cq$take$test;
64 1 = CQ$NEXT$TAKE:
= Procedure (q$ptr) byte external;
65 2 = Declare q$ptr address;
66 2 = end cq$next$take;
67 1 = CQ$Q$INIT:
= Procedure (q$ptr,q$size) external;
68 2 = Declare q$ptr address;
69 2 = Declare q$size byte;
70 2 = end cq$q$init;
71 1 = CQ$ASC$HEX:
= Procedure (q$ptr,msg$ptr) external;
72 2 = Declare (q$ptr,msg$ptr) address;
73 2 = end cq$asc$hex;
74 1 = CQ$CHECKSUM:
= Procedure (q$ptr) byte external;
75 2 = Declare q$ptr address;
76 2 = end cq$checksum;
77 1 = CQ$HEX$ASC:
= Procedure (q$ptr,hex$byte) external;
78 2 = Declare c$ptr address;
79 2 = Declare hex$byte byte;
80 2 = end cq$hex$asc;
81 1 = CQ$MSG$OUT:
= Procedure (msg$size,q$ptr,par$ptr,msg$ptr) external;
82 2 = Declare msg$size byte;

```

```

83 2 = Declare (q$ptr,par$ptr,msg$ptr) address;
84 2 = end cq$msg$out;
85 1 = CQ$GEN$RDY:
    = Procedure (trget,msg$ptr,q$ptr,par$ptr) external;
86 2 = Declare trget byte;
87 2 = Declare (msg$ptr,q$ptr,par$ptr) address;
88 2 = end cq$gen$rdy;
89 1 = CQ$GEN$MSG:
    = Procedure (msg$pointer,trget,msg$ptr,q$ptr,par$ptr,save$flg) external;
90 2 = Declare (trget,save$flg) byte;
91 2 = Declare (msg$pointer,msg$ptr,q$ptr,par$ptr) address;
92 2 = end cq$gen$msg;
93 1 Declare RQL6FX int$exchange$descriptor public;
94 1 Declare CQ$IEX exchange$descriptor public;
95 1 Declare cmrqex exchange$descriptor;
96 1 declare mld$slave$ex exchange$descriptor public;
97 1 Declare cqtime exchange$descriptor public;

98 1 cq$start$msg$351:
    procedure external;
99 2 end cq$start$msg$351;

100 1 cq$init$351:
    procedure external;
101 2 end cq$init$351;

102 1 find$sexch:
    procedure(name$ptr) address external;
103 2 declare name$ptr address;
104 2 end find$sexch;

105 1 cq$startup:
    procedure external;
106 2 end cq$startup;

107 1 declare cq$in$sl queue$format public;
108 1 declare cq$out$sl queue$format public;
109 1 declare msg msg$format;
110 1 declare c$pars par$format public;
111 1 declare fre$mem address;
112 1 declare cqcmdf(5) byte external;

113 1 declare fre$msg based fre$mem structure (
    msg$hdr);

/*
The message which is transmitted between nodes
follows the description below:

The complete message structure is defined as:

STX  TARGET  COMMAND  SEQ  TEXT  CHECKSUM  EOT

```

```

-----
I LF I CC-FF I C-F I LA,NA I n I CC-FF I CR I
-----
0 1 2 3 4 5 6 6+n 7+n 8+n

```

With the exception of the start of text (a line feed) and the end of transmission (a carriage return), all characters transmitted in the message string will consist of printable ASCII characters.

The target field consists of two frames which represent the hex address of the device to which the message is addressed. The protocol allows for up to 256 unique device addresses.

The command field indicates the type of message which is being sent in the network. It consists of a single frame representing a hex number in the range from 0 to FFH. The current message definitions are:

command	label
0	INIT
1	SYNCH
2	DATA
3	READY
4	NOT READY
5	NON SEQ ACK
6	reserved
.	"
.	"
.	"
F	"

This is the main link level RMX/RF communications task which supports slave operations of a single board computer. Several high level functions are supported by this task.

Messages containing data may be sent or received by this task. To send a message to another processor on the communications loop, a message of the format shown is placed into the communications exchange, CQSIEX. When the message has been transmitted, the RAM area in which the message was located will be returned to the free space manager.

Messages may also be received by the board when they are addressed to the communications address assigned to this board. When a message has been received, it will be transferred to a block of RAM obtained from the free space manager and then sent to an exchange which is designated in the name field of the message.

The format for a data message is:

```
msg$hdr targetexchangenam message
```

Data may be up to 250 bytes in length.

A special command, INIT, can be received by the communications driver task which will cause a software reset of the single board computer.

*/

```

114 1  CC$SLAV: Procedure public;
115 2      declare target$exch address;
116 2      declare name address;
117 2      declare msg$ptr address;
118 2      declare ram$size address;
119 2      declare RAM$msg based msg$ptr structure (
          msg$hdr );
120 2      declare req$msg structure (
          msg$hdr,
          msg$length address );
          /* initialize the task at power up */
121 2      call cq$qq$init(.cq$in$sl, 250);
122 2      call cq$qq$init(.cq$out$sl, 250);
123 2      cq$pars.run, cq$pars.stop, cq$pars.que$pt = 0;

          /* build required exchanges */
124 2      call rqcxch(.rq16ex);
125 2      call rqcxch(.cqsiex);
126 2      call rqcxch(.cmrrex);
127 2      call rqcxch(.cqtime);
128 2      call cqinit$351;
129 2      do forever;

          /* wait for a window from the master */
130 3      msg$ptr = rawait (.RQL6EX, 400);

          /*test for good communications with iMOS*/
131 3      if ram$msg.type <> 3 then do;

          /* test for the receipt of a valid message */
133 4      if cq$checksum(.cq$in$sl) = 0
          then do;

          /* convert the message into hex format */
135 5      call cq$asc$hex(.cq$in$sl,.msg);

          /* see if message is for this slave board */
136 5      if msg.trgt = cqslad
          then do;

```

```

138 6  name = .msg.text + 9;
139 6  cqcmdef(0) = cqcmdef(0) and 7fh;

/* handle each message type by case */
do case msg.cmd;

/* case 0, INIT command */
call cqstartup;

/* case 1, SYNCH command */
do;

/* reset counters */
cq$pars.la, cq$pars.na = 0;

/* initialize data queues */
call cq$qs$init (.cq$in$sl, 250);
call cq$qs$init (.cq$out$sl, 250);

/* return non seq ack message */
msg.trgt = 0;
msg.cmd = 5;
call cq$msg$out (0, .cq$out$sl, .cq$

call cq$start$sl;

end;

/* case 2, DATA command */
do;

/* test for correct NA */
if cq$pars.na = msg.seq$na
then do;

/* clear old messages */
msg$ptr = rdcpt(.hold$sl, slave$e
if msg$ptr > 0
then call return$ram(msg$ptr);

/* increment LA counter */
if (cq$pars.la = cq$pars.la + 1)
then cq$pars.la = 0;

/* verify that exchange exists */
TARGET$EXCH = FIND$EXCH(name);
if TARGET$exch > 0
then do;

/* get RAM and store messa

162 10  - ge */
163 10

```

```

164 10      - = .cm$rq$ex;      read$msg.response$exch
165 10      - ext(2);      read$msg.msg$length = msg.t
166 10      - msg);      call RQSEND (.rqfsax,.req$
167 10      - x, 0);      msg$ptr = RQWAIT (.cm$rq$e
168 10      - th;      msg$ptr = req$msg.msg$len
169 10      - g.text(0), msg$ptr);      call move (msg.text(2),.ms
170 10      - msg$ptr);      call RQSEND (target$exch,
171 10      - /      end;
172 9      - /      /* test for data out request *
173 9      - cq$out$ssl,.cq$pars);      msg$ptr = RQACPT (.cq$ix);
175 9      -      if msg$ptr = 0
176 10      - h(msg$ptr + 9)) > 0      then call cq$gen$rdy (0,.msg,.
178 10      - ch, msg$ptr);      else do;
179 11      - 0,.msg,.cq$out$ssl,.cq$pars,0);      if( target$exch:= find$ex
180 11      - e$ex, msg$ptr);      then call rqsend(target$ex
181 11      -      else do;
182 10      -      call cq$gen$msg(msg$ptr,
183 9      -      call rqsend(.hold$sslav
184 8      -      end;
185 9      -      end;
186 9      -      end;
187 9      -      /* if bad na, then retransmit last
188 9      -      else do;
189 10      -      cq$pars.na = msg.seq$na;
190 10      -      msg$ptr = rqacpt(.hold$sslav$aval
191 10      -      if msg$ptr > 0
192 9      -      then do;
193 9      -      call cq$gen$msg(msg$ptr,r,
194 9      -      .msg,.cq$out$ssl,.cq$pars,r);
195 9      -      call rqsend(.hold$sslav$ex
196 9      -      end;
197 9      -      end;
198 9      -      end;
199 9      -      end;
200 9      -      end;
201 9      -      end;
202 9      -      end;

```

```

193 8      /* send message */
194 8      call cq$start$msg$351;
      end;

195 7      /* case 3, RDY command */
      do;

196 8      /* verify na is correct */
      if cq$pars.na = msg.seq$na
      then do;

198 9      /* clear old messages */
      msg$ptr = r$acpt(.hold$slave$e
199 9      - x);
      if msg$ptr > 0
      then call return$ram(msg$ptr);

      /* test for a message waiting
      to be sent */
201 9      msg$ptr = RQACPT (.cq$ix);
202 9      if msg$ptr = 0
      then call cq$gen$rdy (0,.msg,.
      - cq$out$sl,.cq$pars);
204 9      else do;
205 10      if (target$exch:=find$exch
      - (msg$ptr + 9)) > 0
      then call r$send(target$ex
      - ch,msg$ptr);
207 10      else do;
208 11      call cq$gen$msg(msg$ptr
      - r,0,.msg,.cq$out$sl,.cq$pars,0);
209 11      call r$send(.hold$slave$
      - e$ex,msg$ptr);
210 11      end;
211 10      end;

212 9      end;

      /* if bad na, then retransmit last
      - */
213 8      else do;
214 9      cq$pars.na = msg.seq$na;
215 9      msg$ptr = r$acpt(.hold$slave$e
      - x);
216 9      if msg$ptr > 0
      then do;
218 10      call cq$gen$msg(msg$ptr,0,
      - .msg,.cq$out$sl,.cq$pars,0);
219 10      call r$send(.hold$slave$ex
      - ,msg$ptr);
220 10      end;
221 9      end;

      /* start transmission */

```



```

222 8      /* send message */      call cqsstart$msg$351;
223 8      end;

/* case 4, NONRDY command */
224 7      do;end;

/* case 5, NONSEQACK command */
226 7      do;end;

/* case 6, reserved */
228 7      do;end;

/* case 7, reserved */
230 7      do;end;

/* case 8, reserved */
232 7      do;end;

/* case 9, reserved */
234 7      do;end;

/* case A, reserved */
236 7      do;end;

/* case B, reserved */
238 7      do;end;

/* case C, reserved */
240 7      do;end;

/* case D, reserved */
242 7      do;end;

/* case E, reserved */
244 7      do;end;

/* case F, reserved */
246 7      do;end;

248 7      end; /* do case */
249 6      end; /* good target */
250 5      end; /* good checksum */
251 4      end; /* good communications with iMOS */

/* clear out messages if com failure exists */
252 3      else do;
253 4          do while (msg$ptr:=rcapt(.caslex)) > 0;
254 5              call return$ram(msg$ptr);
255 5          end;

256 4          do while (msg$ptr:=rcapt(.hold$slave$ex)) > 0;
257 5              call return$ram(msg$ptr);
258 5          end;

/* set failure indicator */

```

```

259 4      cqcndf(0) = cqcndf(0) or 80h;
260 4      end;

```

```

261 3      ; end; /* do forever */
262 2      end ccslav;
263 1      end ccomslvmodule;

```

```

      byte;
      byte;
      byte;
      byte;
      byte;
      byte;
      byte;
      byte;

```

```

      declare pformal literally 'structure'
      byte;
      byte;
      byte;
      byte;
      byte;
      byte;
      byte;
      byte;

```

```

      declare pformal literally 'structure'
      byte;
      byte;
      byte;
      byte;
      byte;
      byte;
      byte;
      byte;

```

```

      include('b:syncp.txt')

```

```

      ROEND;
      PROCEDURE (EXCHANGEPOINTER, MESSAGEPOINTER) EXTERNAL;
      DECLARE (EXCHANGEPOINTER, MESSAGEPOINTER) ADDRESS;

```

```

      END ROEND;

```

```

      WAIT;
      PROCEDURE (EXCHANGEPOINTER, DELAY) ADDRESS EXTERNAL;
      DECLARE (EXCHANGEPOINTER, DELAY) ADDRESS;

```

```

      END WAIT;

```

```

      ROAPT;
      PROCEDURE (EXCHANGEPOINTER) ADDRESS EXTERNAL;
      DECLARE EXCHANGEPOINTER ADDRESS;

```

```

      END ROAPT;

```

```

      ROIND;
      PROCEDURE (IEDEPTR) EXTERNAL;
      DECLARE IEDEPTR ADDRESS;

```

```

      END ROIND;
      include('b:exch.clt')
      DECLARE EXCHANGEPOINTER LITERALLY 'STRUCTURE'
      MESSAGE ADDRESS;
      MESSAGE ADDRESS;
      TASKHEAD ADDRESS;
      TASKTAIL ADDRESS;

```

```

1      $title('QUEUE INITIALIZATION PROCEDURE')
      QSINITSMODULE: Do;

2  1      declare eom literally 'CDH';
      $include(:fl:commsg.elt)

3  1  =      declare msg$format literally 'structure (
      =          trgt      byte,
      =          cmdnd      byte,
      =          seq$la      byte,
      =          seq$na      byte,
      =          text(250) byte )';

4  1  =      declare queue$format literally 'structure (
      =          end$ptr      byte,
      =          give$ptr      byte,
      =          take$ptr      byte,
      =          data$byte(254) byte )';

5  1  =      declare par$format literally 'structure (
      =          la          byte,
      =          na          byte,
      =          run          byte,
      =          stop         byte,
      =          que$pt       byte )';
      $include(:f0:synch.ext)

6  1  =      RQSEND:
      =          PROCEDURE (EXCHANGES$POINTER,MESSAGES$POINTER) EXTERNAL;
7  2  =          DECLARE (EXCHANGES$POINTER,MESSAGES$POINTER) ADDRESS;
      =

8  2  =          END RQSEND;
      =

9  1  =      RQWAIT:
      =          PROCEDURE (EXCHANGES$POINTER,DELAY) ADDRESS EXTERNAL;
10 2  =          DECLARE (EXCHANGES$POINTER,DELAY) ADDRESS;
      =

11 2  =          END RQWAIT;
      =

12 1  =      RQACPT:
      =          PROCEDURE (EXCHANGES$POINTER) ADDRESS EXTERNAL;
13 2  =          DECLARE EXCHANGES$POINTER ADDRESS;
      =

14 2  =          END RQACPT;
      =

15 1  =      RQISND:
      =          PROCEDURE (IED$PTR) EXTERNAL;
16 2  =          DECLARE IED$PTR ADDRESS;
      =

17 2  =          END RQISND;
      $include(:f0:exch.elt)

18 1  =      DECLARE EXCHANGES$DESCRIPTOR LITERALLY 'STRUCTURE (
      =          MSG$HEAD ADDRESS,
      =          MSG$TAIL ADDRESS,
      =          TASK$HEAD ADDRESS,
      =          TASK$TAIL ADDRESS,

```

```

= ('NEXT$EXCH ADDRESS)';
$include(:fc:msg.elt)
19 1 = DECLARE MSG$HDR LITERALLY '
= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
20 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
= MSG$HDR,
= REMAINDER (1) BYTE)';
21 1 declare timex exchange$descriptor external;
22 1 declare rqlsrx exchange$descriptor external;
23 1 CQ$Q$INIT: Procedure (queue$ptr, queue$size) reentrant pub
- lic;
/*
/* This procedure initializes the queue pointers to
indicate an empty queue.
*/
24 2 Declare queue$ptr address;
25 2 Declare queue$size byte;
26 2 Declare queue based queue$ptr queue$format;
/* set up the size of the queue into the structure */
27 2 queue.END$PTR = queue$size - 1;
/* reset the queue offset pointers */
28 2 queue.GIVE$PTR, queue.TAKE$PTR = 0;
/* return to calling program */
29 2 return;
30 2 end cq$q$init;
31 1 end q$init$module;

```



```

$title('NEXT GIVE QUEUE PROCEDURE, VERSION 2.1')
1      NEXT$GIVE$MODULE: Do;
2      1      declare eom literally 'CDH';
           $include(:fl:commsg.elt)
3      1      =      declare msg$format literally 'structure (
           =      trgt      byte,
           =      cmnd      byte,
           =      seq$la      byte,
           =      seq$na      byte,
           =      text(250) byte )';
4      1      =      declare queue$format literally 'structure (
           =      end$ptr      byte,
           =      give$ptr      byte,
           =      take$ptr      byte,
           =      data$byte(254) byte )';
5      1      =      declare par$format literally 'structure (
           =      la      byte,
           =      na      byte,
           =      ca:      byte,
           =      stop      byte,
           =      que$pt      byte )';
           $include(:ff:synch.ext)
6      1      =      RQSEND:
           =      PROCEDURE (EXCHANGE$PCINTER,MESSAGE$POINTER) EXTERNAL;
7      2      =      DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
           =
8      2      =      END RQSEND;
           =
9      1      =      RQWAIT:
           =      PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
10     2      =      DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
           =
11     2      =      END RQWAIT;
           =
12     1      =      RQACPT:
           =      PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
13     2      =      DECLARE EXCHANGE$POINTER ADDRESS;
           =
14     2      =      END RQACPT;
           =
15     1      =      RQISND:
           =      PROCEDURE (IED$PTR) EXTERNAL;
16     2      =      DECLARE IED$PTR ADDRESS;
           =
17     2      =      END RQISND;
           $include(:ff:exch.elt)
18     1      =      DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
           =      MSG$HEAD ADDRESS,
           =      MSG$TAIL ADDRESS,
           =      TASK$HEAD ADDRESS,
           =      TASK$TAIL ADDRESS,

```

```

= NEXT$EXCH ADDRESS)';
$include(:f0:msg.elt)
19 1 = DECLARE MSG$HDR LITERALLY '
= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
20 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE (
= MSG$HDR,
= REMAINDER(1) BYTE)';

21 1 declare timex exchange$descriptor external;
22 1 declare rqfsrx exchange$descriptor external;

23 1 CQ$NEXT$GIVE: Procedure(Queue$PTR,GIVE$BYTE) byte reentran
- t public;
/*
This procedure places a byte into the queue if room
exists in the queue for that byte of data. If no room
exists, a queue full indicator will be returned by
the procedure.
*/

24 2 Declare Queue$FULL literally 'OFFH';
25 2 Declare Queue$OK literally 'OOFH';

26 2 Declare Queue$PTR address;
27 2 Declare (GIVE$BYTE,RSLT) byte;

28 2 declare queue based queue$ptr queue$format;

/* Test for queue full condition and if it is full,
then insert end of message */

29 2 RSLT = queue$ok;
30 2 If (queue.GIVE$PTR+1 > queue.END$PTR)
then do;
32 3 rslt = queue$full;
33 3 queue.data$byte(queue.give$ptr) = eom;
34 3 end;
35 2 else do;

/* Store the byte into the next queue location
- */

36 2 queue.DATA$BYTE(queue.GIVE$PTR) = GIVE$BYTE;

/* Increment the give pointer */

37 3 If ((queue.GIVE$PTR:=queue.GIVE$PTR+1))

```



```

1      $title('GET CHARACTER FROM QUEUE PROCEDURE')
      NEXT$TAKES$MODULE: Do;

2      1      declare eom literally 'CDH';
      $include(:f1:commsg.elt)

3      1      =      declare msg$format literally 'structure (
      =          trgt      byte,
      =          cmnd      byte,
      =          sec$la      byte,
      =          sec$na      byte,
      =          text(255) byte )';

4      1      =      declare queue$format literally 'structure (
      =          end$ptr      byte,
      =          give$ptr      byte,
      =          take$ptr      byte,
      =          data$byte(254) byte )';

5      1      =      declare par$format literally 'structure (
      =          la      byte,
      =          na      byte,
      =          run      byte,
      =          stop      byte,
      =          que$pt      byte )';
      $include(:f0:synch.ext)

6      1      =      RQSEND:
      =          PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;
7      2      =      DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
      =
8      2      =      END RQSEND;
      =

9      1      =      RQWAIT:
      =          PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
10     2      =      DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
      =
11     2      =      END RQWAIT;
      =

12     1      =      RQACPT:
      =          PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
13     2      =      DECLARE EXCHANGE$POINTER ADDRESS;
      =
14     2      =      END RQACPT;
      =

15     1      =      RQISND:
      =          PROCEDURE (IED$PTR) EXTERNAL;
16     2      =      DECLARE IED$PTR ADDRESS;
      =
17     2      =      END RQISND;
      $include(:f0:exch.elt)

18     1      =      DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
      =          MSG$HEAD ADDRESS,
      =          MSG$TAIL ADDRESS,
      =          TASK$HEAD ADDRESS,
      =          TASK$TAIL ADDRESS,

```



```

= NEXT$EXCH ADDRESS);';
$include(:fc:msg.elt)
19 1 = DECLARE MSG$HDR LITERALLY '
= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
20 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE (
= MSG$HDR,
= REMAINDER(1) BYTE)';
=
21 1 declare tinex exchange$descriptor external;
22 1 declare rqfsrx exchange$descriptor external;
23 1 CQ$NEXT$TAKE: Procedure (queue$ptr) byte reentrant public;
/*
This typed procedure gets the next byte from the
indicated queue and returns it to the calling
program. The queue take pointer is incremented.
*/
24 2 Declare queue$ptr address;
25 2 Declare take$byte byte;
26 2 declare queue based queue$ptr queue$format;
/* store next data byte in queue */
27 2 take$byte = queue.DATAS$BYTE(queue.TAKE$PTR);
/* increment the take pointer to next location */
28 2 If ((queue.TAKE$PTR=queue.TAKE$PTR+1)
> queue.END$PTR)
then queue.TAKE$PTR = 0;
/* return the data byte to caller */
30 2 return take$byte;
31 2 end cq$next$take;
32 1 end next$take$module;

```

```

$title('ASCII to HEX CONVERSION PROCEDURE')
1  ASC$MODULE: Do;

2  1  declare eom literally 'CDH';
    $include(:fl:commsg.elt)
3  1  =  declare msg$format literally 'structure (
    =      trgt      byte,
    =      cmdnd     byte,
    =      seq$la     byte,
    =      seq$na     byte,
    =      text(250) byte )';
    =
4  1  =  declare queue$format literally 'structure (
    =      end$ptr     byte,
    =      give$ptr    byte,
    =      take$ptr    byte,
    =      data$byte(254) byte )';
    =
5  1  =  declare par$format literally 'structure (
    =      la          byte,
    =      na          byte,
    =      run         byte,
    =      stop        byte,
    =      que$pt      byte )';
    $include(:fc:synch.ext)
6  1  =  RQSEND:
    =      PROCEDURE (EXCHANGESPOINTER,MESSAGE$POINTER) EXTERNAL;
7  2  =      DECLARE (EXCHANGESPOINTER,MESSAGE$POINTER) ADDRESS;
    =
8  2  =      END RQSEND;
    =
9  1  =  RQWAIT:
    =      PROCEDURE (EXCHANGESPOINTER,DELAY) ADDRESS EXTERNAL;
10 2  =      DECLARE (EXCHANGESPOINTER,DELAY) ADDRESS;
    =
11 2  =      END RQWAIT;
    =
12 1  =  RQACPT:
    =      PROCEDURE (EXCHANGESPOINTER) ADDRESS EXTERNAL;
13 2  =      DECLARE EXCHANGESPOINTER ADDRESS;
    =
14 2  =      END RQACPT;
    =
15 1  =  RQISND:
    =      PROCEDURE (IED$PTR) EXTERNAL;
16 2  =      DECLARE IED$PTR ADDRESS;
    =
17 2  =      END RQISND;
    $include(:fc:exch.elt)
18 1  =  DECLARE EXCHANGESDESCRIPTOR LITERALLY 'STRUCTURE (
    =      MSG$HEAD ADDRESS,
    =      MSG$TAIL ADDRESS,
    =      TASK$HEAD ADDRESS,
    =      TASK$TAIL ADDRESS,

```

```

= NEXT$EXCH ADDRESS)';
$include(:f0:msg.elt)
19 1 = DECLARE MSG$HDR LITERALLY '
= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
20 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
= MSG$HDR,
= ALIAS(1) BYTE)';
21 1 declare timex exchange$descriptor external;
22 1 declare rqfsrx exchange$descriptor external;
23 1 cq$next$take:
= procedure (q$ptr) byte external;
24 2 declare q$ptr address;
25 2 end cq$next$take;
26 1 CQ$ASC$HEX: Procedure(q$ptr,msg$ptr)reentrant public;
/*
This procedure transforms ASCII data in the input
queue into a hex string in the system message storage
area.
*/
27 2 Declare (nchar,char,n) byte;
28 2 Declare (q$ptr,msg$ptr) address;
29 2 Declare msg based msg$ptr msg$format;
/* throw away the start of message */
30 2 char = cq$next$take(q$ptr);
/* convert message target address */
31 2 char = cq$next$take(q$ptr);
32 2 nchar = cq$next$take(q$ptr);
33 2 if char > 40h
then char = char - 37h;
35 2 else char = char - 30h;
36 2 if nchar > 40h
then nchar = nchar - 37h;
38 2 else nchar = nchar - 30h;
39 2 msg.trgt = shl(char,4) + nchar;
/* convert command byte */
40 2 char = cq$next$take(q$ptr);
41 2 if char > 40h
then char = char - 37h;
43 2 else char = char - 30h;

```

```

44 2      msg.cmd = char;
      /* convert LA sequence */
45 2      char = cq$next$take(q$ptr);
46 2      if char > 40h
47 2      then char = char - 37h;
48 2      else char = char - 30h;
49 2      msg.seq$la = char;
      /* convert NA sequence */
50 2      char = cq$next$take(q$ptr);
51 2      if char > 40h
52 2      then char = char - 37h;
53 2      else char = char - 30h;
54 2      msg.seq$na = char;
      /* do operations until end of message */
55 2      n = 0;
56 2      Do while (char:=cq$next$take(q$ptr)) <> EOM;
      /* get next ASCII character */
57 3      nchar = cq$next$take(q$ptr);
      /* convert to hex format */
58 3      if char > 40h
59 3      then char = char - 37h;
60 3      else char = char - 30h;
61 3      if nchar > 40h
62 3      then nchar = nchar - 37h;
63 3      else nchar = nchar - 30h;
64 3      msg.text(n) = shl(char,4) + nchar;
65 3      n = n + 1;
66 3      end;
67 2      end cq$asc$hex;
68 1      end asc$hex$module;

```



```

1      $title('HEX TO ASCII CONVERSION PROCEDURE')
      HEX$ASC$MODULE: Do;

2      1      'declare eom literally 'CDH';
      $include(:fl:commsg.elt)

3      1      =      declare msg$format literally 'structure (
      =          trgt      byte,
      =          cmdnd      byte,
      =          seq$la      byte,
      =          seq$na      byte,
      =          text(250) byte )';

4      1      =      declare queue$format literally 'structure (
      =          end$ptr      byte,
      =          give$ptr      byte,
      =          take$ptr      byte,
      =          data$byte(254) byte )';

5      1      =      declare par$format literally 'structure (
      =          la      byte,
      =          na      byte,
      =          run      byte,
      =          stop      byte,
      =          que$pt      byte )';
      $include(:ff:synch.ext)

6      1      =      RQSEND:
      =          PROCEDURE (EXCHANGES$POINTER, MESSAGE$POINTER) EXTERNAL;
7      2      =          DECLARE (EXCHANGES$POINTER, MESSAGE$POINTER) ADDRESS;
      =

8      2      =          END RQSEND;
      =

9      1      =      RQWAIT:
      =          PROCEDURE (EXCHANGES$POINTER, DELAY) ADDRESS EXTERNAL;
10     2      =          DECLARE (EXCHANGES$POINTER, DELAY) ADDRESS;
      =

11     2      =          END RQWAIT;
      =

12     1      =      RQACPT:
      =          PROCEDURE (EXCHANGES$POINTER) ADDRESS EXTERNAL;
13     2      =          DECLARE EXCHANGES$POINTER ADDRESS;
      =

14     2      =          END RQACPT;
      =

15     1      =      RQISND:
      =          PROCEDURE (IED$PTR) EXTERNAL;
16     2      =          DECLARE IED$PTR ADDRESS;
      =

17     2      =          END RQISND;
      $include(:ff:exch.elt)

18     1      =      DECLARE EXCHANGES$DESCRIPTOR LITERALLY 'STRUCTURE (
      =          MSG$HEAD ADDRESS,
      =          MSG$TAIL ADDRESS,
      =          TASK$HEAD ADDRESS,
      =          TASK$TAIL ADDRESS,

```

```

= 'NEXT$EXCH ADDRESS)';
$include(:ff:msg.elt)
19 1 = DECLARE MSG$HDR LITERALLY '
= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
20 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
= MSG HDR,
= REL: (1) BYTE)';
21 1 declare timex exchange$descriptor external;
22 1 declare rqfsrx exchange$descriptor external;
23 1 cq$next$give:
  procedure(que$ptr,data$byte) byte external;
24 2 declare que$ptr address;
25 2 declare data$byte byte;
26 2 end cq$next$give;
27 1 CQ$HEX$ASC: Procedure (q$ptr,hex$byte) reentrant public;
/*
  This procedure is used to convert a hex byte into two
  ASCII characters and store them into the next two loc-
  ations of the Q$OUT data area.
*/
28 2 Declare (high$byte,hex$byte,low$byte,status) byte;
29 2 Declare q$ptr address;
30 2 Declare q$OUT based q$ptr queue$format;
  /* separate low and high order bits */
31 2 If (high$byte:=(shr(hex$byte,4) and CFH)) > 9
  then high$byte = high$byte + 37H;
33 2 else high$byte = high$byte + 3FH;
34 2 If (low$byte:=(hex$byte and CFH)) > 9
  then low$byte = low$byte + 37H;
36 2 else low$byte = low$byte + 30H;
  /* store ASCII conversions into the queue */
37 2 status = cq$next$give(.Q$OUT, high$byte);
38 2 status = cq$next$give(.Q$OUT, low$byte);
39 2 return;
40 2 end cq$hex$asc;
41 1 end hex$asc$module;

```

```

1      $title('CHECKSUM CALCULATION PROCEDURE')
CHECKSUM$MODULE: Do;

2      1      declare com literally 'CDH';
$include(:f1:commsg.elt)

3      1      =      declare msg$format literally 'structure (
=      trgtr      byte,
=      cmnd      byte,
=      seq$la      byte,
=      seq$na      byte,
=      text(250) byte )';

4      1      =      declare queue$format literally 'structure (
=      end$ptr      byte,
=      give$ptr      byte,
=      take$ptr      byte,
=      data$byte(254) byte )';

5      1      =      declare par$format literally 'structure (
=      la      byte,
=      na      byte,
=      run      byte,
=      stop      byte,
=      que$pt      byte )';
$include(:f0:synch.ext)

6      1      =      ROSEND:
=      PROCEDURE (EXCHANGESPOINTER, MESSAGESPOINTER) EXTERNAL;
7      2      =      DECLARE (EXCHANGESPOINTER, MESSAGESPOINTER) ADDRESS;
=
8      2      =      END ROSEND;
=
9      1      =      RQWAIT:
=      PROCEDURE (EXCHANGESPOINTER, DELAY) ADDRESS EXTERNAL;
10     2      =      DECLARE (EXCHANGESPOINTER, DELAY) ADDRESS;
=
11     2      =      END RQWAIT;
=
12     1      =      RQACPT:
=      PROCEDURE (EXCHANGESPOINTER) ADDRESS EXTERNAL;
13     2      =      DECLARE EXCHANGESPOINTER ADDRESS;
=
14     2      =      END RQACPT;
=
15     1      =      RQISND:
=      PROCEDURE (IED$PTR) EXTERNAL;
16     2      =      DECLARE IED$PTR ADDRESS;
=
17     2      =      END RQISND;
$include(:f0:exch.elt)

18     1      =      DECLARE EXCHANGESDESCRIPTOR LITERALLY 'STRUCTURE (
=      MSG$HEAD ADDRESS,
=      MSG$TAIL ADDRESS,
=      TASK$HEAD ADDRESS,
=      TASK$TAIL ADDRESS,

```

```

= NEXT$EXCH ADDRESS)";
$include(:ff:msg.elt)
19 1 = DECLARE MSG$HDR LITERALLY '
= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
20 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
= MSG$HDR,
= REMAINDER(1) BYTE)'

21 1 declare timex exchange$descriptor external;
22 1 declare rqlsrx exchange$descriptor external;
23 1 CQ$CHECKSUM: Procedure(q$ptr) byte reentrant public;

/*
This procedure is used to determine the checksum of
a message which has been received and stored into
the Q$in buffer. A checksum will be calculated on all
characters beginning at the start of text and
continuing until the checksum bytes are encountered.
This value will be tested with the value transmitted
and if they agree, a value of zero will be returned
to the calling program. If not in agreement, a
non-zero value will be returned.
*/

24 2 Declare (ptr,end$ptr,q$ptr) address;
25 2 Declare (checksm,string$sum,char,n) byte;
26 2 Declare last$chars(3) byte;

27 2 declare q$in based q$ptr queue$format;

/* get queue pointer values for reference */

28 2 PTR = Q$IN.take$ptr;
29 2 END$PTR = Q$IN.end$ptr;

/* initialize checksum value */

30 2 checksm = 0;

/* compute checksum of string */

31 2 char = Q$IN.data$byte(ptr);
32 2 do while char <> EOM;
33 3 checksm = checksm + char;
34 3 if ptr = end$ptr
then ptr = 0;
else ptr = ptr + 1;
35 3 char = Q$IN.data$byte(ptr);
36 3 char = Q$IN.data$byte(ptr);
37 3 char = Q$IN.data$byte(ptr);
38 3 end;

```



```

/* last three characters in the message are
not included in the checksum, so they must
be subtracted....*/

/* first remember the last queue location */
39 2    if ptr = end$ptr
41 2    then char = 0;
    else char = ptr+1;

42 2    do n = 0 to 2;
43 3    checksum = checksum - (last$chars(n) :=
    Q$IN.data$byte(ptr));
44 3    if ptr = 0
    then ptr = end$ptr;
45 3    else ptr = ptr - 1;
47 3    end;
48 2    checksum = checksum + last$chars(0);

/* convert transmitted checksum into a hex value */
49 2    do n = 1 to 2;
50 3    if last$chars(n) > 40H
    then last$chars(n) = last$chars(n) - 37H;
52 3    else last$chars(n) = last$chars(n) - 30H;
53 3    end;
54 2    string$sum = shl(last$chars(2),4) + last$chars(1);

/* test for validity of transmission */

55 2    if string$sum = checksum
    then return 0;

/* if bad checksum, clear data queue of message */
57 2    else Q$IN.take$ptr = char;
58 2    return -1;

59 2    end co$checksum;
60 1    end checksum$module;

```

```

1      $title('GENERATE READY MESSAGE PROCEDURE')
      GENSRDYSMODULE: Do;

2      1      declare eom literally 'CDH';
      $include(:f1:commsg.elt)

3      1      =      declare msg$format literally 'structure (
      =      trg      byte,
      =      cmd      byte,
      =      seq$la     byte,
      =      seq$na     byte,
      =      text(250) byte )';

4      1      =      declare queue$format literally 'structure (
      =      end$ptr     byte,
      =      give$ptr    byte,
      =      take$ptr     byte,
      =      data$byte(254) byte )';

5      1      =      declare par$format literally 'structure (
      =      la          byte,
      =      na          byte,
      =      run          byte,
      =      stop         byte,
      =      que$pt       byte )';
      $include(:f0:synch.ext)

6      1      =      RQSEND:
      =      PROCEDURE (EXCHANGESPOINTER,MESSAGE$POINTER) EXTERNAL;
7      2      =      DECLARE (EXCHANGESPOINTER,MESSAGE$POINTER) ADDRESS;
      =

8      2      =      END RQSEND;
      =

9      1      =      RQWAIT:
      =      PROCEDURE (EXCHANGESPOINTER,DELAY) ADDRESS EXTERNAL;
10     2      =      DECLARE (EXCHANGESPOINTER,DELAY) ADDRESS;
      =

11     2      =      END RQWAIT;
      =

12     1      =      RQACPT:
      =      PROCEDURE (EXCHANGESPOINTER) ADDRESS EXTERNAL;
13     2      =      DECLARE EXCHANGESPOINTER ADDRESS;
      =

14     2      =      END RQACPT;
      =

15     1      =      RQISND:
      =      PROCEDURE (IED$PTR) EXTERNAL;
16     2      =      DECLARE IED$PTR ADDRESS;
      =

17     2      =      END RQISND;
      $include(:f0:exch.elt)

18     1      =      DECLARE EXCHANGES$DESCRIPTOR LITERALLY 'STRUCTURE (
      =      MSG$HEAD ADDRESS,
      =      MSG$TAIL ADDRESS,
      =      TASK$HEAD ADDRESS,
      =      TASK$TAIL ADDRESS,

```

```

= NEXTSEXCH ADDRESS);
#include(:f0:msg.clt)
19 1 = DECLARE MSG$HDR LITERALLY '
= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
20 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
= MSG$HDR,
= REMAINDER(1) BYTE)';
=
21 1 declare timex exchange$descriptor external;
22 1 declare rqfsrx exchange$descriptor external;
23 1 cq$msg$out:
= procedure (msg$size,q$ptr,par$ptr,msg$ptr) external;
24 2 declare msg$size byte;
25 2 declare (q$ptr,par$ptr,msg$ptr) address;
26 2 end cq$msg$out;
27 1 CQ$GEN$RDY: Procedure (trget,msg$ptr,q$ptr,par$ptr) reentr
- ant public;

/*
This procedure generates a "ready" message onto the
communication network. The target address is passed
as the parameter in the call.

On entry to the procedure:
trget is a byte which contains the address
of the target node.
msg$ptr is an address which points to
the RAM work area.
q$ptr is an address which points to
the output queue.
par$ptr is an address which points to
the communication flags.
*/

28 2 Declare trget byte;
29 2 Declare rdy$msg structure (
= TARGET byte,
= COMAND byte,
= SEQ$LA byte,
= SEQ$NA byte )
= data ( 0,3,0,0 );
30 2 declare (msg$ptr,q$ptr,par$ptr) address;
31 2 declare msg based msg$ptr msg$format;

/* move format into message block */

```

```

32  2      call move (4, .rdy$msg, msg$ptr);
          /* insert current parameters */
33  2      msg.trgt = trgt;
          /* send message */
34  2      call cq$msg$out (0, c$ptr, par$ptr, msg$ptr);
35  2      return;
36  2      end cq$gen$rdy;
37  1      end gen$rdy$module;

```



```

1      $title('DATA MESSAGE GENERATOR PROCEDURE')
GEN$MSC$MODULE: Do;

2      1      declare eom literally 'CDH';
$include(:fl:commsg.elt)

3      1      =      declare msg$format literally 'structure (
=          trgt      byte,
=          cmd      byte,
=          seq$la    byte,
=          seq$na    byte,
=          text(250) byte )';

4      1      =      declare queue$format literally 'structure (
=          end$ptr    byte,
=          give$ptr   byte,
=          take$ptr   byte,
=          data$byte(254) byte )';

5      1      =      declare par$format literally 'structure (
=          la         byte,
=          na         byte,
=          run        byte,
=          stop       byte,
=          que$pt     byte )';
$include(:f0:synch.ext)

6      1      =      RQSEND:
=          PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;
7      2      =          DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
=
8      2      =          END RQSEND;
=
9      1      =      RQWAIT:
=          PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
10     2      =          DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
=
11     2      =          END RQWAIT;
=
12     1      =      RQACPT:
=          PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
13     2      =          DECLARE EXCHANGE$POINTER ADDRESS;
=
14     2      =          END RQACPT;
=
15     1      =      RQISND:
=          PROCEDURE (IED$PTR) EXTERNAL;
16     2      =          DECLARE IED$PTR ADDRESS;
=
17     2      =          END RQISND;
$include(:f0:exch.elt)
18     1      =      DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
=          MSG$HEAD ADDRESS,
=          MSG$TAIL ADDRESS,
=          TASK$HEAD ADDRESS,
=          TASK$TAIL ADDRESS,

```

```

= NEXT$EXCH ADDRESS);
$include(:ff:msg.elt)
19 1 = DECLARE MSG$HDR LITERALLY '
= LINK ADDRESS,
= LENGTH ADDRESS,
= TYPE BYTE,
= HOME$EXCHANGE ADDRESS,
= RESPONSE$EXCHANGE ADDRESS';
=
20 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
= MSG$HDR,
= REMAINDER(1) BYTE)';
21 1 declare timex exchange$descriptor external;
22 1 declare rqsrx exchange$descriptor external;
23 1 cq$msg$out:
= procedure (msg$size, a$ptr, par$ptr, msg$ptr) external;
24 2 declare msg$size byte;
25 2 declare (a$ptr, par$ptr, msg$ptr) address;
26 2 end cq$msg$out;
27 1 CQ$GEN$MSG: Procedure (msg$pointer, trget, msg$ptr, a$ptr,
- par$ptr, save$flg) reentrant public;
/*
This procedure generates a data message and places
it onto the system communications network.

On entry to the procedure:
msg$pointer is an address which points to
the data to be transmitted.
trget is an byte which contains the
address of the target node.
msg$ptr is an address which points to
the RAM work area.
a$ptr is an address which points to
the output queue.
par$ptr is an address which points to
the communications flags.
*/
28 2 Declare (msg$pointer, msg$ptr, a$ptr, par$ptr, ram$size) a
- ddress;
29 2 Declare (trget, save$flg) byte;
30 2 Declare msg based msg$ptr msg$format;
31 2 Declare data$msg structure (
target byte,
comand byte,
seq$LA byte,
seq$NA byte,
text byte )
data (0,2,0,0,0);

```

```

32  2      declare data$block based msg$pointer structure (
           msg$hdr );
           /* move message format into message buffer */
33  2      call move (5, .data$msg, msg$ptr);

           /* insert target address */
34  2      msg.trgt = trgt;
           /* append data to communication message */
35  2      call move (data$block.length, msg$pointer, .msg.text(0)
           - (n));
           /* transmit message onto network */

36  2      ram$size = data$block.length;
37  2      call cq$msg$out (ram$size, q$ptr, par$ptr, msg$ptr);
           /* return memory to free space manager */

38  2      if save$flg
           then do;
39  3          if ram$size mod 4 > 0
           then data$block.length = data$block.length + (4 - (r
           am$size mod 4));
42  3      call RQSEND (.rqf$rx, msg$pointer);
43  3      end;

44  2      return;
45  2      end cq$gen$msg;
46  1      end gen$msg$module;

```

```

$title('iSBX 351 Communications Driver, Version 1.2')
$nointvector
1  CQS351: Do;

/*****

This module contains the physical interface for
support of the Intel iSBX 351 communications expansion
board. The board is inserted into socket J6 and has a
base address of F0H with the interrupt from the
receiver strapped to the interrupt level 6. The
transmitter interrupt is wired to interrupt level 5.

Multi-drop communication options are supported by
the physical software package.

*****/

$include (:fc:exch.elt)
2  1  =  DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
    =  MSG$HEAD ADDRESS,
    =  MSG$TAIL ADDRESS,
    =  TASK$HEAD ADDRESS,
    =  TASK$TAIL ADDRESS,
    =  NEXT$EXCH ADDRESS)';
$include (:fc:ied.elt)
3  1  =  DECLARE INT$EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
    =  MESSAGE$HEAD ADDRESS,
    =  MESSAGE$TAIL ADDRESS,
    =  TASK$HEAD ADDRESS,
    =  TASK$TAIL ADDRESS,
    =  EXCHANGE$LINK ADDRESS,
    =  LINK ADDRESS,
    =  LENGTH ADDRESS,
    =  TYPE BYTE)';
4  1  =  Declare RQL6EX int$exchange$descriptor external;
5  1  =  Declare RQL7EX int$exchange$descriptor external;

$include (:fl:comm$g.elt)
6  1  =  declare msg$format literally 'structure (
    =  trgt      byte,
    =  cmd      byte,
    =  seq$la   byte,
    =  seq$na   byte,
    =  text(250) byte)';

7  1  =  declare queue$format literally 'structure (
    =  end$ptr   byte,
    =  give$ptr  byte,
    =  take$ptr  byte,
    =  data$byte(254) byte)';

8  1  =  declare par$format literally 'structure (

```



```

=      la      byte,
=      na      byte,
=      run     byte,
=      stop    byte,
=      queue$pt byte )';
9      1      Declare CQINSL queue$format external;
10     1      Declare CQOUTSL queue$format external;
11     1      Declare CQPARS par$format external;

      $include (:fl:comcom.ext)
12     1      = CQ$NEXT$GIVE:
      =      Procedure (q$ptr,give$byte) byte external;
13     2      =      Declare q$ptr address;
14     2      =      Declare give$byte byte;
15     2      =      end cq$next$give;
16     1      = CQ$TAKE$TEST:
      =      Procedure (q$ptr) byte external;
17     2      =      Declare a$ptr address;
18     2      =      end cq$take$test;
19     1      = CQ$NEXT$TAKE:
      =      Procedure (q$ptr) byte external;
20     2      =      Declare q$ptr address;
21     2      =      end cq$next$take;
22     1      = CQ$Q$INIT:
      =      Procedure (q$ptr,q$size) external;
23     2      =      Declare q$ptr address;
24     2      =      Declare q$size byte;
25     2      =      end cq$q$init;
26     1      = CQ$ASC$HEX:
      =      Procedure (q$ptr,msg$ptr) external;
27     2      =      Declare (q$ptr,msg$ptr) address;
28     2      =      end cq$asc$hex;
29     1      = CQ$CHECKSUM:
      =      Procedure (q$ptr) byte external;
30     2      =      Declare q$ptr address;
31     2      =      end cq$checksum;
32     1      = CQ$HEX$ASC:
      =      Procedure (q$ptr,hex$byte) external;
33     2      =      Declare a$ptr address;
34     2      =      Declare hex$byte byte;
35     2      =      end cq$hex$asc;
36     1      = CQ$MSC$OUT:
      =      Procedure (msg$size,q$ptr,par$ptr,msg$ptr) external;
37     2      =      Declare msg$size byte;
38     2      =      Declare (a$ptr,par$ptr,msg$ptr) address;
39     2      =      end cq$msg$out;
40     1      = CQ$GEN$RDY:
      =      Procedure (trget,msg$ptr,a$ptr,par$ptr) external;
41     2      =      Declare trget byte;
42     2      =      Declare (msg$ptr,a$ptr,par$ptr) address;
43     2      =      end cq$gen$rdy;
44     1      = CQ$GEN$MSG:
      =      Procedure (msg$pointer,trget,msg$ptr,a$ptr,par$ptr,save$flg) external;
45     2      =      Declare (trget,save$flg) byte;

```

```

46 2 =      Declare (msg$pointer,msg$ptr,q$ptr,par$ptr) address;
47 2 =      end cq$gen$msg;
      $include (:fc:synch.ext)
48 1 =      RQSEND:
      =      PROCEDURE (EXCHANGESPOINTER,MESSAGESPOINTER) EXTERNAL;
49 2 =      DECLARE (EXCHANGESPOINTER,MESSAGESPOINTER) ADDRESS;
      =
50 2 =      END RQSEND;
      =
51 1 =      RQWAIT:
      =      PROCEDURE (EXCHANGESPOINTER,DELAY) ADDRESS EXTERNAL;
52 2 =      DECLARE (EXCHANGESPOINTER,DELAY) ADDRESS;
      =
53 2 =      END RQWAIT;
      =
54 1 =      RQACPT:
      =      PROCEDURE (EXCHANGESPOINTER) ADDRESS EXTERNAL;
55 2 =      DECLARE EXCHANGESPOINTER ADDRESS;
      =
56 2 =      END RQACPT;
      =
57 1 =      RQISND:
      =      PROCEDURE (IED$PTR) EXTERNAL;
58 2 =      DECLARE IED$PTR ADDRESS;
      =
59 2 =      END RQISND;
      $include (:fc:intrpt.ext)
60 1 =      RQENDI:
      =      PROCEDURE EXTERNAL;
      =
61 2 =      END RQENDI;
      =
62 1 =      RQELVL:
      =      PROCEDURE (LEVEL) EXTERNAL;
63 2 =      DECLARE LEVEL BYTE;
      =
64 2 =      END RQELVL;
      =
65 1 =      RQDLVL:
      =      PROCEDURE (LEVEL) EXTERNAL;
66 2 =      DECLARE LEVEL BYTE;
      =
67 2 =      END RQDLVL;
      =
68 1 =      RQSETV:
      =      PROCEDURE (PROC,LEVEL) EXTERNAL;
69 2 =      DECLARE PROC ADDRESS;
70 2 =      DECLARE LEVEL BYTE;
      =
71 2 =      END RQSETV;
      =
72 1 =      RQSETP:
      =      PROCEDURE (PROC,LEVEL) EXTERNAL;
73 2 =      DECLARE PROC ADDRESS;
74 2 =      DECLARE LEVEL BYTE;

```

```

75 2 =      END RQSETP;

76 1      Declare EOM literally 'CDH';
          /* defines end of message */
77 1      Declare RTS literally '00100000b';
          /* ready to send to USART */
78 1      Declare RXE literally '00000100b';
          /* ready to receive to USART */
79 1      Declare DTR literally '00000010b';
          /* data set ready to USART */
80 1      Declare TXEN literally '00000001b';
          /* transmit enable to USART */
81 1      Declare badint byte;
          /* flag for interrupt 5 */

Seject
/*****

      This procedure initializes the hardware required to
      operate the iSPX 351 expansion board.

      *****/

82 1      CQ$INIT$351:
          Procedure public;
83 2      Declare n byte;

          /* initialize the timer/counters */
84 2      output(0fbh) = 0b6h;          /* select counter 2 */
85 2      output(0fah) = 32;          /* lsb for 2400 baud */
86 2      output(0fah) = 0;          /* msb for 2400 baud */

          /* initialize the USART */
87 2      output(0flh) = 80h;          /* clear out garbage */
88 2      output(0flh) = 0;          /* ... more garbage */
89 2      output(0flh) = 40h;          /* reset the USART */
90 2      output(0flh) = 0ceh;        /* 8 bit, no parity, 2 st
- op */
91 2      output(0flh) = 26h;          /* enable receive mode */

          /* tell the nucleus the vector location */

92 2      CALL rqsetv(.cqmivt$351, 6);
93 2      call rasetv(.send$char$351, 5);
94 2      call rqlvl( 5);
95 2      call rqlvl(6);
96 2      return;
97 2      end cq$init$351;

Seject
/*****

```

This procedure stores a character from the USART into the receiver data input queue.

```

*****/
98 1  CQMIVT$351:
    Procedure interrupt 6 public;
99 2  Declare (GIVE$STATUS,CHAR) byte;

    /* get character from USART */
100 2  char = input(0F0h) and 07Fh;
101 2  give$status = cq$next$give(.cqinsl, char);
102 2  if char = eom
    then call rqisnd(.rql6ex);
104 2  else call rqendi;

105 2  end cqmivt$351;
Seject
/*****
    This procedure sends out the next character to the
    selected USART device. It will disable the interrupt
    when all desired characters have been transmitted.
*****/

106 1  SEND$CHAR$351:
    Procedure interrupt 5 public;
107 2  Declare char byte;

    /*testing if interrupt is really for 5 and not noise*/
108 2  if badint > 0 then do;

    /* enable drivers at beginning of message */
110 3  If not cqpars.run
    then do;
112 4      cqpars.run = 1;
113 4      cqpars.stop = 0;
114 4  end;

    /* disable drivers at end of message */
115 3  If cqpars.stop
    then do;
117 4      cqpars.run = 0;
118 4      Do while (input(0F1h) and 4) = 0;
119 5      end;
120 4      badint = 0;
121 4      output(0F1h) = 26h;
122 4  end;

    /* if message is in progress, send next character */
    else do;
123 3      char = cq$next$take(.cqouts1);
124 4      do while (input(0F1h) and 4) = 0;
125 5      end;
126 5      output(0F0h) = char;
127 4

    /* test for end of message */
128 4  if (char and 0fh) = eom

```



```

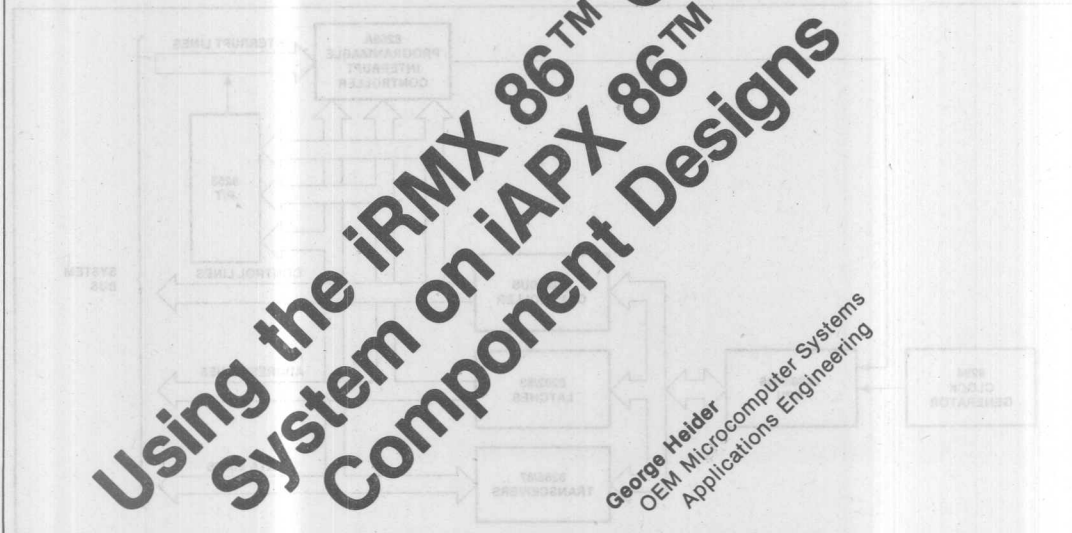
130      4          then cqpars.stop = 1;
131      4          else cqpars.stop = 0;
132      3          end;
133      2          /* re-enable interrupts */
134      2          call rqendi;
135      2          return;
136      2          end send$char$351;
137      2          $reject
138      2          /*****
139      2          This procedure begins the transmission to a selected node.
140      2          *****/
141      1          CQ$START$MSG$351:
142      1          Procedure public;
143      2          badint = 0ffh;
144      2          output(0flh) = 25h;
145      2          return;
146      2          end cq$start$msg$351;
147      1          end cq$351;
148
149      /* enable driver at beginning of message */
150      if not cqpars.run
151      then do;
152          cqpars.run = 1;
153          cqpars.stop = 1;
154      end;
155
156      /* disable driver at end of message */
157      if cqpars.stop
158      then do;
159          cqpars.run = 0;
160          Do while (input(0flh) and 0) = 0;
161          end;
162          badint = 0;
163          output(0flh) = 25h;
164      end;
165
166      /* if message is in progress, send next character */
167      else do;
168          char = cqnxt$take(,cqnxt$);
169          Do while (input(0flh) and 0) = 0;
170          end;
171          output(0flh) = char;
172
173      /* test for end of message */
174      if (char and 0flh) = com

```

The operating system interfaces allow the use of standard software tools such as debugger, Operating system also provide decreased debugging time and increased reliability through error checking and error handling. Perhaps most important, the expertise gained can be carried on to new designs based on the operating system.

Operating systems have generally been described as large and complex, with rigid system requirements. Users have found it difficult to tailor a system to their needs or to use the operating system on more than one hardware configuration. System software has been associated in large pieces or as a whole with the system configuration choice in either hardware or software. Those systems shall enough to use on component designs have lacked extensibility to larger, more complex designs.

The Intel iRMX 86™ Operating System offers users of component hardware all benefits of operating systems while imposing few hardware restrictions. Minimum hardware requirements include 1.5K RAM memory, enough RAM or EPROM memory to hold the iRMX 86™ and the application code, and a handful of integrated circuits. The circuitry includes Intel iAPX 86™, Intel 8232/83™ Latches for bus address lines, an Intel 8232/83™ Programmable Interval Timer, and an Intel 8232/83™ Programmable Interrupt Controller. Larger system buses will also require an Intel 8232/83™ Bus Controller and Intel 8232/83™ Transceivers for data lines. This basic hardware system is shown in Figure 1.



Using the iRMX 86™ Operating System on iAPX 86™ Component Designs

George Heider
OEM Microcomputer Systems
Applications Engineering

INTRODUCTION

A specification system based on a custom hardware design will typically perform faster and require less hardware than if it were implemented with "off the shelf" circuit boards. However, these advantages are counteracted by the disadvantages of custom designs, with one of the largest drawbacks being the custom software involved. This software is often unique to the application and specific to the hardware design, requiring a substantial and increasing percentage of the development and testing effort. The cost is multiplied by the need for software tools, standards, and maintenance procedures necessary for each application. In addition, the software is often costly to move to the hardware.

January 1981

new applications on hardware. All of these disadvantages can be significantly reduced by using a modular, standardized operating system. The operating system provides a higher level interface to the system hardware. The hardware characteristics common to most applications, such as memory management and interrupt handling, are handled by the operating system rather than the application software. The operating system provides scheduling and synchronization for multiple functions, allowing application code to be written in larger units or modules. The operating system interface is more standardized than the interface to the hardware components. This allows the application software to be independent of the hardware. The application can be in charge of hardware. The application can be implemented and debugged on the hardware. The software is then easily moved to the hardware for testing.

Figure 1. Basic Hardware System for the iRMX 86™ Operating System

INTRODUCTION

An application system based on a custom hardware design will typically perform faster and require less hardware than if it were implemented with "off the shelf" circuit boards. However, these advantages are countered by the disadvantages of custom designs, with one of the largest drawbacks being the custom software required. This software is often unique to the application and specific to the hardware design, requiring a significant and increasing percentage of the development schedule and expense. The cost is multiplied by the need for software tools, standards, and maintenance developed specifically for each application. In addition, much of the application software cannot be used for new applications or hardware. All of these disadvantages can be significantly reduced by using a modular, standardized operating system.

The operating system provides a higher level interface to the system hardware. The hardware characteristics common to most applications, such as memory management and interrupt handling, are handled by the operating system rather than the application software. The operating system provides scheduling and synchronization for multiple functions, allowing application code to be written in independent pieces or modules. The operating system interface can be more standardized than the interface to the hardware components. This allows the application software to be more independent of changing hardware. The application code can be initially implemented and debugged on proven hardware. The software is then easily moved to the final hardware configuration for testing.

The operating system interfaces allow the use of standard software tools, such as debuggers. Operating systems also provide decreased debugging time and increased reliability through error checking and error handling. Perhaps most important, the expertise gained can be carried on to new designs based on the operating system.

Operating systems have generally been described as large and complex, with rigid system requirements. Users have found it difficult to tailor a system to their needs or to use the operating system on more than one hardware configuration. System software has been accepted in large pieces or as a whole, with few system configuration choices in either hardware or software. Those systems small enough to use on component designs have lacked extendibility to larger, more complex designs.

The Intel iRMX 86 Operating System offers users of component hardware all benefits of operating systems while imposing few hardware restrictions. Minimum hardware requirements include 1.8K RAM memory, enough RAM or EPROM memory to hold the Nucleus and the application code, and a handful of integrated circuits. The circuits are an Intel iAPX 86 or iAPX 88 Central Processing Unit, an Intel 8284 Clock Generator, Intel 8282/83 Latches for bus address lines, an Intel 8253 Programmable Interval Timer, and an Intel 8259A Programmable Interrupt Controller. Larger system busses will also require an Intel 8288 Bus Controller and Intel 8286/87 Transceivers for data lines. This basic hardware system is shown in Figure 1.

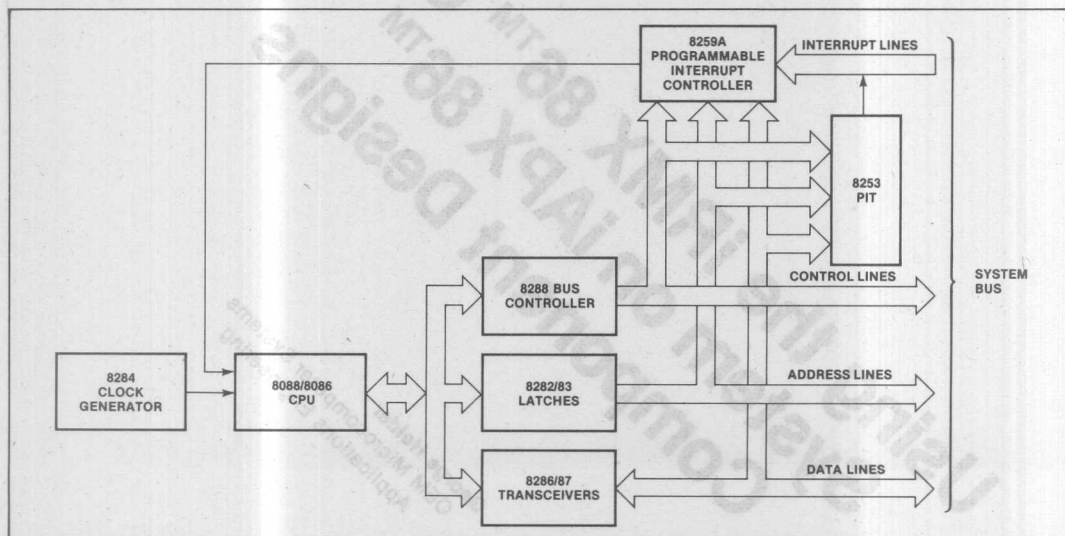


Figure 1. Basic Hardware System for the iRMX 86™ Operating System

Users with a wide range of applications will find the iRMX 86 Operating System allows them to implement a corresponding range of capabilities, from a minimum iRMX 86 Nucleus to a high level human interface. A complete iRMX 86 Operating System includes extensive I/O capabilities, debugger, application loader, bootstrap loader, and integrated user functions. This flexibility allows one operating system to be used for many projects, minimizing software learning curves for new applications.

This note discusses a relatively small standalone spectrum analysis system based on a subset of the iRMX 86 Nucleus. The intent of the note is to demonstrate advantages of using operating systems in hardware component designs. An overview of operating system functions is given first as background information. Readers familiar with operating systems may wish to skip this section. The overview is followed by a summary of the iRMX 86 Operating System. The summary is brief, as only the iRMX 86 Nucleus is used in this application. A detailed discussion may be found in Application Note AP-86, "Using the iRMX 86 Operating System," and iRMX 86 System Manuals.

The spectrum analysis system is described after the summary. The system requirements, design and implementation are detailed. The system software is discussed next, followed by configuration and hardware implementation. A summary completes the application note text. Partial code listings of the system software are included in the appendices.

OPERATING SYSTEMS FUNCTIONAL OVERVIEW

Operating system software manage initialization, resources, scheduling, synchronization, and protection of tasks or functions within the system, as well as providing facilities for maintenance, debugging, and growth. In general, operating systems support many of the following:

Multiprogramming

Multiprogramming provides the capability for two or more programs to share the system hardware, after being developed and implemented independently. Within the environment of an operating system, the programs are called jobs. Jobs include system resources, such as memory, in addition to the actual program code. Multiprogramming allows jobs that are required only during development, such as debuggers, to run in the target system. When development is completed, these jobs are removed from the final system without affecting the integrity of the remaining jobs.

Multitasking

Multitasking allows functions within a job to be handled by separate tasks. This is particularly valuable when a job is responsible for multiple asynchronous events or activities. One task can be assigned to each event or activity. Tasks are the functional members of the system, executing within the bounds of a job environment. Program code for a multitasking system is modular, with well-defined interfaces and communication protocols. The modular boundaries serve several important purposes. The code for each module can be generated and tested independent of the other modules. In addition, the boundaries confine errors, speeding debugging and simplifying maintenance.

Growth

The modular independence that results from multiprogramming and multitasking gives users the ability to efficiently create new applications by adding functions to old software. Applications can be tailored to specific needs by integrating new modules with previously-written general support code. If care is taken in system design, functions can be added in the field. Documentation for the older software can be carried on to the new applications. This growth path will save completely re-writing expensive custom software for each new application.

Scheduling

Even though a system has multiple jobs and tasks, only one task is actually running on the central processor at any single point in time. Scheduling provides a means of predicting and controlling the selection of the running task from the tasks that are ready to run. Basic scheduling methods include preemptive priority, non-preemptive priority, time-slice, and round-robin. Batch systems often use non-preemptive priority scheduling, in which the highest priority job gets control of the central processor and runs to completion. Preemptive scheduling is typically used in real-time or event-driven systems, where dedicated, quick response is the main concern. A higher-priority waiting task that becomes ready to run will preempt the lower-priority running task. Priority may be either set at task creation (static) or modified during running of the task (dynamic).

Time-slice and round-robin scheduling are used in multiuser or multitask systems that share processing resources and have limits on maximum execution time. Time-slice scheduling gives each task or job a fixed slice of dedicated processor time. Round-robin gives each task or job a turn at using the processor. The time available during the turn depends on system load and task priority.

Communication and Synchronization

Jobs and tasks in a multiprogramming and multitasking environment require a structured means of communication. This communication may be necessary to synchronize processes or to pass data between processes. Two means of providing communication are mailboxes and semaphores. Mailboxes are an exchange place for system messages. The messages may include data or provide access to other system objects, including other mailboxes. Tasks can send objects to a mailbox or wait for objects from a mailbox. Generally, the task has the option of waiting for a specified period of time for the message. The wait time may range from zero for a task that requires immediate response to infinite time for a task that must have a message to continue processing. Multiple mailboxes are used to synchronize multiple tasks.

A communication flow using mailboxes is shown in Figure 2. In this example, the sending task sends a message to a mailbox and specifies a return mailbox. The sending task then waits at the return mailbox. The receiving task obtains the message from the sending mailbox and sends a message to the return mailbox. The first task obtains the second message from the return mailbox, synchronizing the two tasks and passing data.

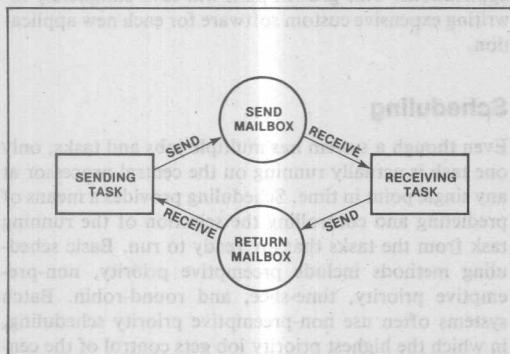


Figure 2. Intertask Communications with Mailboxes

If process synchronization is the only requirement, semaphores may be used. Semaphores function like mailboxes except that no data is passed through the semaphore. Instead, semaphores contain "units," with the meaning of the units defined by the sending and receiving tasks. A one unit semaphore may be used as a flag to synchronize the tasks. Multiple unit semaphores can be used for resource control. For example, if tasks require reusable data buffers, a semaphore may be defined as the allocator of the available data buffers. Each unit in the semaphore will represent one available buffer. When a task requires buffers to continue, the task will wait at the semaphore until enough units (representing

buffers) are available. The waiting task will receive the units, use the buffers, and return the units (still representing buffers) to the semaphore. Other tasks that require buffers will also have to wait at the semaphore until enough buffers are available.

Resource Management

The operating system is the central guardian of system resources, specifically read/write memory. The memory is made known to the Nucleus at initialization. The Nucleus then gives pieces or segments of the memory to tasks as they request it. This allows the tasks to have no initial knowledge of the actual location and size of system memory. Tasks can share memory if they desire, but the Nucleus allocates memory to each task individually, preventing the tasks from using each others memory. In addition, tasks return memory to the Nucleus when they are through with it, allowing memory to be reused.

Other system resources, such as I/O devices, will also be scheduled by the operating system. The operating system is responsible both for the efficient use of these resources and for providing the tasks with a large measure of independence from the actual I/O hardware requirements. The system may require many types of I/O devices, such as disk drives, tape drives, and printers. I/O is more efficiently accomplished if the operating system provides both asynchronous and synchronous I/O operations. Synchronous operations are those the task starts and waits for completion, doing no other work until the I/O is complete. Asynchronous operations are started by the task, but the actual I/O can take place while the task is doing other work. The overlapped operation of asynchronous I/O provides more user control of the I/O operation at the expense of a more complicated user interface.

Interrupt Management

Real-time software is tightly coupled to hardware functions by interrupts. Interrupts provide rapid notification that the hardware needs attention. The software must respond quickly without corrupting the system environment. System integrity is preserved by preempting the lower priority operating task, saving the task environment, processing the interrupt (including communicating the results to other tasks if necessary), restoring the environment of the operating task, and continuing. All of this must occur in an orderly and efficient manner. The interrupt management of the operating system is responsible for directly interacting with the system hardware that detects interrupts. The interrupt tasks can be ignorant of the detailed interrupt hardware, providing only the system actions to service the event that caused the interrupt.

Initialization

Operating systems create and manage jobs and tasks at initialization as well as run time. Initialization generally must be done in a specific sequence which will depend on the environment existing at that time. An abortive initialization environment may require an orderly shut-down of the system. The operating system has the capability for managing these situations, including communications, access to system resource information, and displaying status of the initialization actions.

Debugging

The system debugger is a window into the internal structures of the operating system. Debuggers allow data structures and memory to be examined, breakpoints to be set, and the user to be notified of abnormal conditions. The debugger may have symbolic debugging, in which system objects such as addresses, tasks, jobs, mailboxes, and memory locations can be assigned names. This gives greater flexibility and accuracy during debugging. The debugger may not be necessary in the final system, so the debugger is often a separate system job. This allows removal of the debugger with no effect on the remaining jobs.

Debugging will be aided if the operating system verifies the parameters required by system actions and also returns status for the requested action. Parameter verification is particularly valuable for new program code in a developing system. Status results other than success are abnormal or exception conditions. Exception conditions may include insufficient memory for the request, invalid input data, inoperative I/O devices, an invalid request for an action, or a request for an invalid action. The operating system may have an exception handler for these conditions and may allow the debugger to be used as the exception handler. The development process will be more efficient if detection of exception conditions takes place for all levels of system actions, from initialization of jobs and tasks to requests for memory or status.

Higher Level Functions

With the continuing increase in system complexity, more operating systems are providing higher level functions. These functions may include advanced I/O file management, operator console, spooling operations, telecommunications support, multiuser support and access to system resources of increasing size and complexity. Only the largest operating systems provide all of these capabilities, but users of component hardware must be careful their system will integrate higher level functions that may be required in future applications.

Extendability

In order to provide general purpose support, operating systems must be extendible. New applications may require data structures or system actions not available with the present operating system. The system must be able to integrate these new structures and actions, supporting them in the same manner as existing functions. Choosing an operating system requires a large commitment, both in initial expense and system architecture. Extendibility provides assurance the operating system chosen will not provide built-in obsolescence of that commitment.

iRMX 86™ OPERATING SYSTEM ARCHITECTURE

Layers

The iRMX 86 Operating System architecture is shown in Figure 3. It includes the Nucleus, Basic I/O System, Extended I/O System, Applications Loader, and Human Interface. These major portions of the operating system are designed as layers. Each layer may be added to previous layers as application needs grow. Lower layers may be used without upper layers. All layers may reside in programmable read only memory. Applications have access to all portions of the system, from the Nucleus to all outer layers.

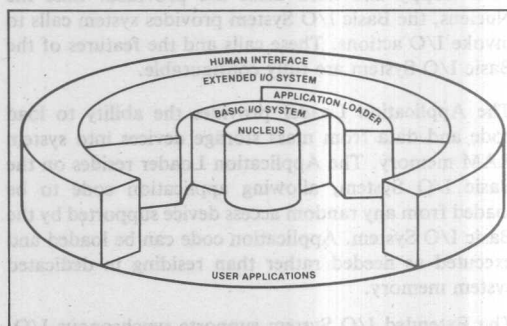


Figure 3. Architecture of the iRMX 86™ Operating System

The Nucleus is the heart of the system. It includes support for multiprogramming, multitasking, communications, synchronization, scheduling, resource management, extendibility, interrupt handling, and error detection. The Nucleus may be considered as an extended layer of the underlying hardware, giving the hardware system management functions and making the software independent of the detailed hardware. The system environment, including resources, priorities, and placement of program code, is made known to the Nucleus at system initialization. All requests for memory, communication, and creation of basic data structures must

go through the Nucleus. These requests are made by system calls, which are comparable to subroutine calls for system actions.

All higher level functions of the iRMX 86 Operating System are built around a core of the Nucleus. Although outer layers may require a substantial number of the system functions included in the Nucleus, the Nucleus itself is configurable on a call-by-call basis. "Configurable" means the Nucleus may be altered so it contains code only for those functions required by the application. Certain features, such as parameter validation and exception handling, are also configurable. Features and system calls may be included for development and excluded from the final system, giving a Nucleus tailored for each level of application development.

The Basic I/O System is the first layer above the Nucleus. The Basic I/O System provides asynchronous I/O support and format independent manipulation of data. Multiple file types are supported, including Stream, Named, and Physical files. Stream files are internal files for transferring large amounts of data between jobs or tasks. Named files include data files of varying sizes and directories for those files. Named files are designed for random access disk storage. Physical files consider the entire device to be one file. Physical files are primarily used to transfer data to and from printers, tape drivers, and terminals. Device drivers for both floppy and hard disks are provided. Like the Nucleus, the Basic I/O System provides system calls to invoke I/O actions. These calls and the features of the Basic I/O System are fully configurable.

The Application Loader provides the ability to load code and data from mass storage devices into system RAM memory. The Application Loader resides on the Basic I/O System, allowing application code to be loaded from any random access device supported by the Basic I/O System. Application code can be loaded and executed as needed rather than residing in dedicated system memory.

The Extended I/O System supports synchronous I/O, automatic buffering, and logical names. Synchronous I/O provides a simplified user interface for I/O actions. Automatic buffering improves I/O efficiency by overlapping I/O and application operations wherever possible. Logical name support allows applications to access files with a user-selected name, aiding I/O device independence.

The Human Interface uses all lower layers, forming a high level man-machine interface for user program invocation, command parsing, and file utilities. The primary purpose of the Human Interface is to support the addition of interactive commands. The Human Interface is the basis for pass-through language support and multiple user systems.

Configurability

Configurability means the iRMX 86 Operating System can be changed to include only the system calls and features pertinent to the system under development. Smaller applications start with only the iRMX 86 Nucleus. A subset of Nucleus calls, described later in this application note, provide the basis for management of jobs, tasks, memory, interrupts, communication and synchronization, and support for the Debugger and Terminal Handler.

All systems calls may also use parameter validation as a configuration option. Parameter validation verifies that system calls reference correct system objects before the requested action is performed. During debugging and in hostile environments, validation provides error detection for each system call. This error detection does add some overhead to the calls. Debugged application jobs can perform more efficiently without the validation, while new code can use parameter validation to speed development.

Once errors are detected, there are two means available to handle error recovery. The task can either use the status information to perform error recovery actions or the recovery actions may be performed by a specialized error handling program called an exception handler. Applications may use the Debugger as an exception handler, or use one implemented by the application. There are two classes of errors that may cause control to be given to an exception handler; avoidable errors, such as programmer errors, and unavoidable errors, such as insufficient memory. Exception handlers can be selected to receive control for either or both classes.

Support Functions

The iRMX 86 Operating System includes a Debugger, a Terminal Handler, a Bootstrap Loader, and a Patch Facility. The Debugger examines system objects, using execution and exchange (mailbox and semaphore) breakpoints, symbolic debugging, and exception handling. The Terminal Handler provides a line-editing, mailbox-driven CRT communications capability. The Bootstrap Loader is a fully configurable loader for bootstrap loading on reset or command, from any specific random access device. The Patch Facility gives the capability of patching iRMX 86 Object Code in the field.

Object Orientation

The iRMX 86 Operating System is based on a set of system data structures called objects. These objects include jobs, tasks, mailboxes, semaphores, segments, and regions. Users may also define application-specific objects. Object architecture includes the objects, their parameters, and the functions allowed with the objects.

Object orientation is a formal, hardware-independent definition of hardware-dependent system structures that are currently used by most applications. For example, without object orientation, memory is reserved in advance for system buffers. The application code knows buffer sizes and locations. If buffer requirements grow, requiring a new memory layout, much of the application code will change to accommodate the new buffer sizes and locations. Using object orientation, the application requests a segment (buffer) of a particular size when the buffer is needed. The Nucleus allocates the memory and returns a segment object to the application. If the application needs a larger buffer, it returns the old segment and requests a new one of a larger size. The application obtains more buffers by making requests for more segments. If the hardware changes, the iRMX 86 Operating System is made aware of the changes. The application code uses the same system calls to request and return the segment objects regardless of the hardware configuration.

Objects are provided for modular environments (jobs), application code functions (tasks), communication (mailboxes), synchronization (semaphores), memory (segments), and mutual exclusion (regions). Objects are fundamentally a set of standard interfaces between application code and the iRMX 86 Operating System. The standard interfaces have three primary benefits:

- 1) First, objects provide structures, such as tasks or segments, that are common to all applications. The structures form the basis for a standard set of system calls that make the interface between the application and the operating system more consistent and easier to learn. These calls allow applications to create more objects (segments for buffers, for example), delete them, change them, and inspect them. Development engineers can use their knowledge of the objects on many applications, rather than just the one under development. The common objects allow a common system debugger to be used. The debugger will work for all applications, letting engineers concentrate their efforts on the application itself rather than designing and implementing custom debugging tools.
- 2) Second, the standard characteristics of the object allow consistent error detection and handling. Requests to alter or use objects can be checked for validity before the Nucleus actually performs the request. Errors can be classed as common to all objects or specific to certain objects, giving more precise error information for effective error handling and faster debugging.
- 3) Third, the object interface will be preserved on future releases of the iRMX 86 Operating System. Current application code can be split into independent modules. Future applications can use the modules for

common functions, preserving the investment in application software.

Task Scheduling

The Nucleus controls task scheduling by task priority and task state. Task priority is specified when the task is created. The priority can be altered dynamically. Tasks are classified into one of five classes: Running, Ready, Asleep, Suspended, or Asleep-Suspended. Tasks that have not been created are considered to be non-existent. The State Transition Diagram is shown in Figure 4.

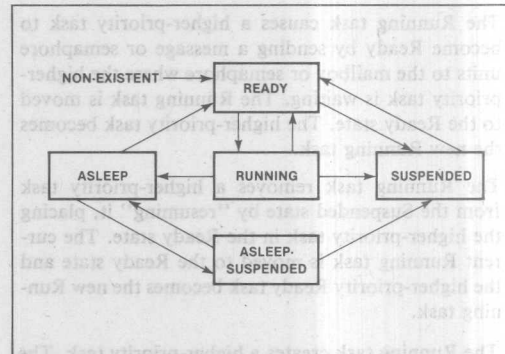


Figure 4. State Transition Diagram

Only one task is in the Running state. This task has control of the central processor. The Ready state is occupied by those tasks that are ready to run but have lower priority than the Running task. The Asleep state is occupied by tasks waiting for a message, semaphore units, availability of a requested resource, an interrupt, or for a requested amount of time to elapse. A task can specify the amount of time it will allow itself to spend in the Asleep state, but tasks in the Suspended state must be "resumed" by other tasks. The Suspended state is useful when situations require firm scheduling beyond the control provided by priority and system resource availability. Examples of these situations are system emergencies, controlling tasks in the Ready state for application-dependent scheduling algorithms, and guaranteeing a fixed initialization or shut-down sequence. If another task "suspends" a task already in the Asleep state, the sleeping task goes to the Asleep-Suspended state. This task will enter the Suspended state if the sleep-causing condition is satisfied. The task will go to the Asleep state from the Asleep-Suspended state if it is resumed before the sleep-causing condition is removed. If a task enters the Ready state and has higher priority than the present Running task, the Ready task is given control of the CPU. Control is transferred to another task only when:

- 1) The Running task makes a request that cannot immediately be filled. The Running task is moved to the

Asleep state. The highest-priority Ready task becomes the Running task.

- 2) An interrupt occurs, causing a higher-priority task to become Ready. The current Running task goes to the Ready state, allowing the higher-priority task to become the Running task.
- 3) The Running task causes a higher-priority task to become Ready by releasing the resource for which the higher-priority task is waiting. The current Running task goes to the Ready state. The higher-priority task becomes the Running task.
- 4) The Running task causes a higher-priority task to become Ready by sending a message or semaphore units to the mailbox or semaphore where the higher-priority task is waiting. The Running task is moved to the Ready state. The higher-priority task becomes the new Running task.
- 5) The Running task removes a higher-priority task from the Suspended state by "resuming" it, placing the higher-priority task in the Ready state. The current Running task is moved to the Ready state and the higher-priority Ready task becomes the new Running task.
- 6) The Running task creates a higher-priority task. The new task goes to the Ready state. The current Running task is moved to the Ready state and the higher-priority Ready task becomes the new Running task.
- 7) The Running task places itself in the Suspended state. The highest-priority Ready task becomes the new Running task.
- 8) The Running task places itself in the Asleep state. The highest-priority Ready task becomes the new Running Task.
- 9) The Running task deletes itself, becoming Non-existent. The highest-priority Ready task will be the new Running task.

System Hardware Requirements

The iRMX 86 Operating System will run on any system that meets the following minimum hardware requirements:

- 1) An iAPX 86 or iAPX 88 Central Processing Unit.
- 2) An Intel 8253 Programmable Interval Timer to provide the system clock.
- 3) An Intel 8259A Programmable Interrupt Controller.
- 4) Enough hardware to provide a system clock and bus interfaces. This may be supplied by the Intel 8284 Clock Generator, Intel 8288 Bus Controller, Intel 8282/8283 Latches, and Intel 8286/8287 Transceivers.

5) The following RAM:

- a. 1024 bytes from 0 to 1024 for software interrupt pointers (the interrupt vector).
 - b. 800 bytes for Nucleus data.
 - c. Enough RAM for the application data, code, and system objects.
- 6) Enough EPROM or RAM to hold the required parts of the iRMX 86 Operating System and the application code.

The Intel iSBC 86/12A Single Board Computer more than meets these minimum requirements. A block diagram of the board is shown in Figure 5. Note in addition to the timer and interrupt controller the board contains an 8251A USART, an 8255 parallel I/O interface, a MULTIBUS™ interface, four sockets for up to 16K bytes of EPROM, and 32K bytes of dual-ported RAM. Even though a user may be developing a custom board for his application, it is recommended that initial system development be accomplished using the iSBC 86/12A Single Board Computer. This will provide a known hardware environment to simplify debugging. In addition, the development hardware system can be adapted to changing application needs by adding Intel MULTIBUS compatible boards to the iSBC 86/12A Single Board Computer. After the software is fully debugged, the application can be moved to the final custom hardware design.

APPLICATION EXAMPLE

A spectrum analyzer is the subject of this application. The analyzer displays the frequency spectrum of an analog signal on a general purpose CRT terminal. The user has control over input signal bandwidth, averaging, and continuous analysis. A fast Fourier transform (FFT) program is used to obtain frequency data from samples of analog data. Fourier transforms provide useful frequency analysis, but the large processing requirements of Fourier transforms have restricted their use. Fast Fourier transforms take advantage of the repetitive nature of the Fourier calculations, allowing the Fourier transforms to be completed significantly faster. The FFT used in this application note is known as "time decomposition with input bit reversal."¹ Sixteen-bit integer samples of the input signal are placed in frames, with each frame holding 128 complex points. An averaged power spectrum is calculated to sum and square the FFT values, yielding 64 32-bit power spectrum values. These values are displayed on a standard CRT terminal.

1. S. O. Stearns, *Digital Signal Analysis*, Hayden Book Co., Rochelle Park, NJ, 1975.

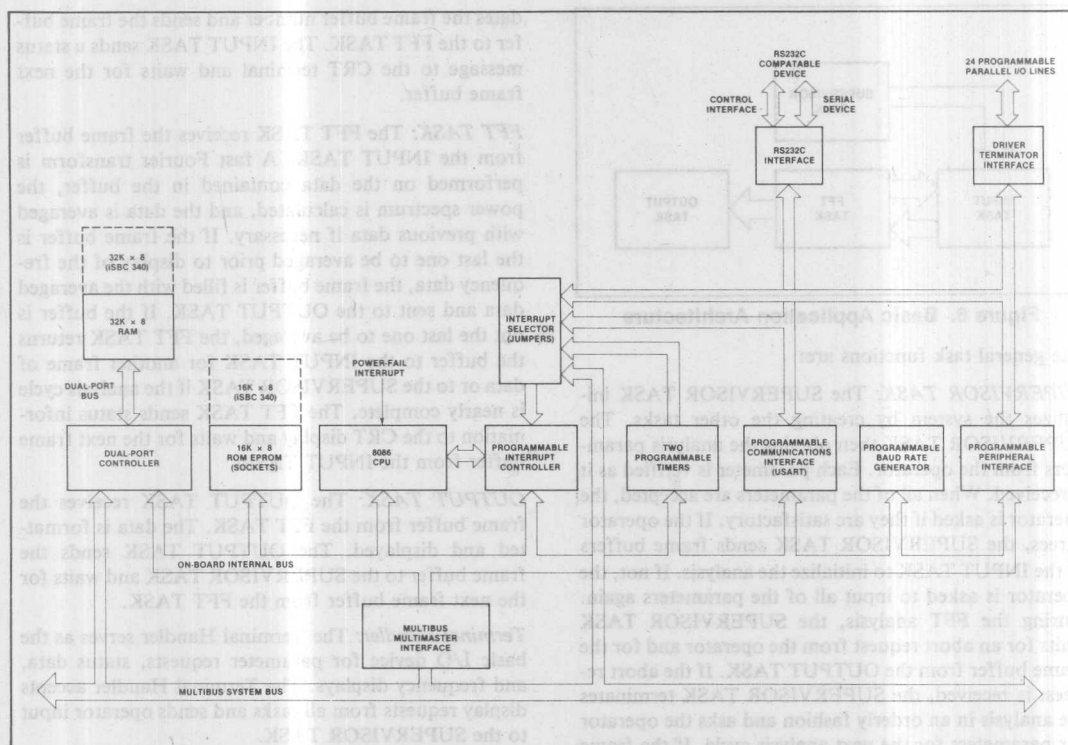


Figure 5. iSBC 86/12A™ Single Board Computer Block Diagram

The FFT algorithm may be applied wherever frequency analysis of an analog signal is required. Medical applications for FFTs include EEG analysis, blood flow analysis, and analysis of other low-frequency body signals. Industrial uses are production line testing, wear analysis, frequency signature monitoring, analysis in noisy or hostile environments, and vibration analysis. Other applications could cover remote reduction of analog data, frequency correlation, and process control. For this application note, the actual use of the FFT is secondary to its existence as a modular, CPU-intensive task in a general purpose system.

The overall application system characteristics are the following:

- 1) A user-selectable input signal bandwidth of 120 Hz, 600 Hz, 1200 Hz, 6000 Hz, or 12,000 Hz.
- 2) The option of averaging frames of samples. The averaging is user selectable, with options of 1 (no averaging), 2, 4, 8, 16, or 32 frames averaged per CRT display.
- 3) The capability, also user selectable, of repeating the analysis cycle continuously.

- 4) User capability to abort the analysis.
- 5) Twelve-bit input sample resolution.
- 6) A minimum of hardware requirements, including no more than 32K bytes of EPROM memory and 16K bytes of RAM memory.
- 7) A standard character screen CRT for output.
- 8) A multitasking structured design that will use a subset of the iRMX 86 Nucleus and exhibit modular application code, formal interfaces, and self-documentation.

System Design

To begin the design, the application is broken up into functional modules, much the same as a hardware block diagram. A SUPERVISOR TASK initializes the system, accepts operator parameters, starts the analysis cycle, and stops the processing upon cycle completion or operator request. An INPUT TASK samples the data and places it in a buffer. An FFT TASK receives the buffer and processes the data. An OUTPUT TASK displays the data received from the FFT TASK. This structure is shown in Figure 6.

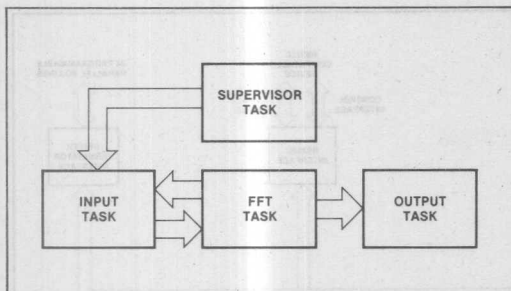


Figure 6. Basic Application Architecture

The general task functions are:

SUPERVISOR TASK: The SUPERVISOR TASK initializes the system by creating the other tasks. The SUPERVISOR TASK then obtains the analysis parameters from the operator. Each parameter is verified as it is received. When all of the parameters are accepted, the operator is asked if they are satisfactory. If the operator agrees, the SUPERVISOR TASK sends frame buffers to the INPUT TASK to initialize the analysis. If not, the operator is asked to input all of the parameters again. During the FFT analysis, the SUPERVISOR TASK waits for an abort request from the operator and for the frame buffer from the OUTPUT TASK. If the abort request is received, the SUPERVISOR TASK terminates the analysis in an orderly fashion and asks the operator for parameters for the next analysis cycle. If the frame buffer is received from the OUTPUT TASK and continuous analysis has been selected, the SUPERVISOR TASK sends the frame buffer to the INPUT TASK to start the next cycle. If the frame buffer is received from the OUTPUT TASK and continuous analysis has not been selected, the current analysis is complete and the SUPERVISOR TASK asks for new parameters.

INPUT TASK: The INPUT TASK receives the frame buffer from the SUPERVISOR TASK. The input signal is sampled according to the analysis parameters. The actual sample rates are calculated as follows:

- 1) Multiply the highest frequency of interest by two to obtain the Nyquist sampling rate.
- 2) Invert this value to obtain time between samples.
- 3) Scale the value by 60/64. The CRT display is limited to 64 columns. The scaling maps sample values to columns 1 to 60 rather than 1 to 64, giving a more readable x-axis label and display. This method yields sample times of 3.9 milliseconds, 781 microseconds, 390 microseconds, 78 microseconds, and 39 microseconds for frequencies of 120 Hz, 600 Hz, 1200 Hz, 6000 Hz, and 12,000 Hz.

The INPUT TASK samples the data at the required interval and places the samples in the frame buffer. When the frame buffer is full, the INPUT TASK up-

dates the frame buffer number and sends the frame buffer to the FFT TASK. The INPUT TASK sends a status message to the CRT terminal and waits for the next frame buffer.

FFT TASK: The FFT TASK receives the frame buffer from the INPUT TASK. A fast Fourier transform is performed on the data contained in the buffer, the power spectrum is calculated, and the data is averaged with previous data if necessary. If the frame buffer is the last one to be averaged prior to display of the frequency data, the frame buffer is filled with the averaged data and sent to the OUTPUT TASK. If the buffer is not the last one to be averaged, the FFT TASK returns the buffer to the INPUT TASK for another frame of data or to the SUPERVISOR TASK if the analysis cycle is nearly complete. The FFT TASK sends status information to the CRT display and waits for the next frame buffer from the INPUT TASK.

OUTPUT TASK: The OUTPUT TASK receives the frame buffer from the FFT TASK. The data is formatted and displayed. The OUTPUT TASK sends the frame buffer to the SUPERVISOR TASK and waits for the next frame buffer from the FFT TASK.

Terminal Handler: The Terminal Handler serves as the basic I/O device for parameter requests, status data, and frequency displays. The Terminal Handler accepts display requests from all tasks and sends operator input to the SUPERVISOR TASK.

The basic functions of the various tasks in the application have been defined, but system integration has not been discussed. Synchronization of the tasks, scheduling, resource management, mapping to hardware, interrupt handling, and system interfaces have been omitted. No debugging functions have been defined. It is clear the system implementation is just started. The iRMX 86 Nucleus will provide all of the system integration "glue" the application requires, allowing application programmers to concentrate on the actual functional code. In order to use this "glue," the application must be divided into jobs and tasks.

Jobs and Tasks

The iRMX 86 Operating System architecture defines jobs as separate environments within which tasks operate. These separate environments allow each job to function with no knowledge of other system jobs. There are two jobs in this application, the Debugger job and the Application job.

The jobs contain functional portions or working programs called tasks. The Application job contains the INIT TASK, SUPERVISOR TASK, INPUT TASK, FFT TASK, OUTPUT TASK, and INTERRUPT TASK. The Debugger job contains the Debugger task and the Terminal Handler task. Tasks provide the ap-

plication goals of modularity, resource constraint boundaries, and functional independence. The structure of the development system is shown in Figure 7.

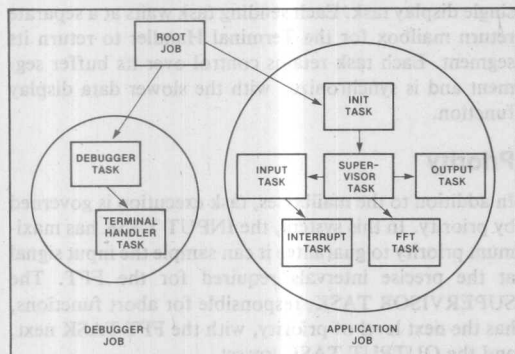


Figure 7. Development System Job Structure

The Debugger job is included only for development. When development is completed, the Debugger job is removed from the system. A Terminal Handler job containing only the Terminal Handler task is substituted in place of the Debugger job. The application code is not changed. This structure is shown in Figure 8.

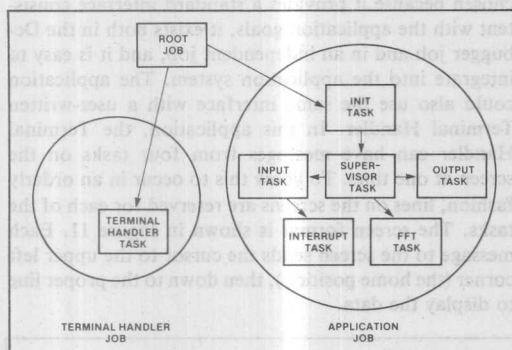


Figure 8. Final System Job Structure

Interfaces and Synchronization

Now that the system is made of jobs and tasks, the primary need is to synchronize the tasks and provide communication interfaces. This will be handled by mailboxes. The messages sent via the mailboxes will be segments, which are pieces of memory allocated by the Nucleus. The frame buffers sent from task to task are these segments. The buffer segments contain all the analysis parameters in addition to the data samples. Communication with the Terminal Handler task is also accomplished by mailboxes, but with a different buffer format. The Terminal Handler format has fields for the operation requested (read or write), the number of characters the task wishes to read or write, the number

of characters the Terminal Handler did read or write, a status field for the operation, and the actual data. The buffer format is shown in Figure 9. Figure 12 contains the Terminal Handler segment format.

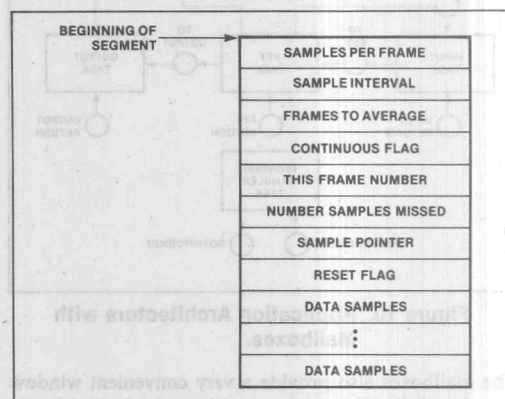


Figure 9. Frame Buffer Format

Mailboxes are used to pass the buffer segment from task to task. Tasks can send segments to mailboxes, or receive segments from mailboxes. If there is no segment at a mailbox, a task can elect to wait for the segment, with the wait duration ranging from zero to forever. This provides the simplest system synchronization — each task, upon initialization, waits at its input mailbox for a frame buffer segment. When the task receives the segment, it processes the data, sends the segment on to the mailbox for the next task, and returns to its own mailbox to wait for the next frame buffer. The system is synchronized and controlled by the availability of frame buffers and task priority. It should be clear that multiple frame buffers segments provide overlapped processing, with the segments ultimately “piling up” at the slowest task in the chain. This loose coupling arrangement allows tasks to have radically different execution times. For example, the INPUT TASK has an input sampling time that ranges from 0.6 seconds to 6.3 milliseconds, a range of 100 to 1, and the system requires no special synchronization or scheduling to accommodate this range.

The mailbox interfaces, shown in Figure 10, serve several other important purposes. First, they provide the standardized interface that is a goal of this application. The set of mailboxes and the two buffer segments form all inter-task interfaces. Each task uses only a few mailboxes, making it easy to add or remove tasks by adding or removing mailboxes. The system could easily be expanded to include data reduction tasks, data correlation tasks, or to substitute different tasks for any of the present ones. Dummy tasks were used for real tasks during development to verify overall system execution before the actual tasks were available.

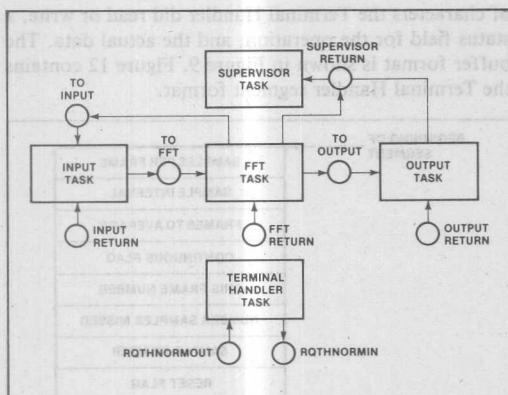


Figure 10. Application Architecture with Mailboxes.

The mailboxes also provide a very convenient window into the application system processing for both debugging and aborting the current cycle. The Debugger can set breakpoints at mailboxes to allow users to examine the frame buffers as they progress from task to task. The Debugger can examine buffer data and control the processing cycle. Tasks wait at the mailboxes in a queue that is either priority or first-in-first-out (FIFO) based. The inter-task mailbox queues are priority based, which means the higher priority SUPERVISOR TASK can intercept segments at the mailboxes ahead of the lower priority waiting tasks, and abort the analysis by removing all of the buffer segments. This method of aborting requires no knowledge of the internal processing of the tasks, making it universally applicable to all the tasks.

A return mailbox may be specified when a segment is sent to a mailbox. The receiving task may send status information, a different segment, or the original segment to the return mailbox. The Terminal Handler will return

the buffer segment sent to it if a return mailbox is specified. This is used to synchronize the tasks with the Terminal Handler and to allow multiple tasks to use a single display task. Each sending task waits at a separate return mailbox for the Terminal Handler to return its segment. Each task retains control over its buffer segment and is synchronized with the slower data display function.

Priority

In addition to the mailboxes, task execution is governed by priority. In this system, the INPUT TASK has maximum priority to guarantee it can sample the input signal at the precise intervals required for the FFT. The SUPERVISOR TASK, responsible for abort functions, has the next level of priority, with the FFT TASK next, and the OUTPUT TASK lowest.

APPLICATION IMPLEMENTATION

Display Functions

All application tasks use the iRMX 86 Terminal Handler as an output device. The Terminal Handler is chosen because it provides a standard interface consistent with the application goals, it exists both in the Debugger job and in an independent job, and it is easy to integrate into the application system. The application could also use the same interface with a user-written Terminal Handler. In this application, the Terminal Handler can have messages from four tasks on the screen at one time. To allow this to occur in an orderly fashion, lines on the screens are reserved for each of the tasks. The screen format is shown in Figure 11. Each message to the screen sends the cursor to the upper left corner (the home position), then down to the proper line to display the data.

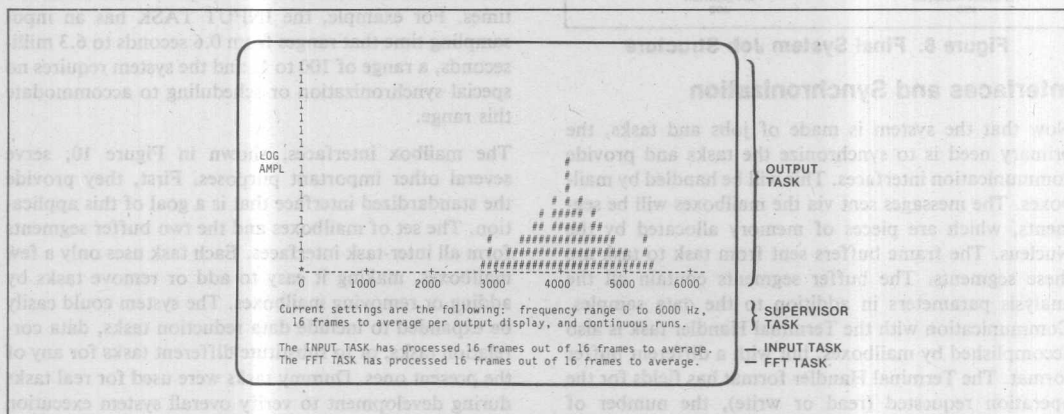


Figure 11. CRT Screen Display Format

Cataloging

To aid the debugging process, all system objects, such as mailboxes, segments, and tasks, are cataloged in the directory of the SUPERVISOR TASK. The catalog entries are user-selected, 12-character names. The Debugger can display this directory, giving easy access to objects to aid symbolic debugging. If other tasks know the proper directory and the 12-character name, the tasks can look up the objects in the directory and obtain access to them. This is the method used to find the Terminal Handler mailboxes. For objects that are cataloged only to aid debugging, the system calls that catalog the objects are removed from the code when debugging is complete.

Application Code

The code listings for the SUPERVISOR TASK and the INPUT TASK are included in appendices to this note. Code listings for other tasks are not included, but they are available from the Intel Insite software library. The following discussions reference line numbers in the listings included in this note. The references begin with a first letter for the appendix section (A or B), followed by the actual line number (A.220, for example).

SUPERVISOR TASK

Code listings for the SUPERVISOR TASK are in Appendix A. The actual SUPERVISOR TASK procedure begins at line A.550. After initializing internal buffers and mailboxes (A.551, A.518–A.549), the Supervisor sends an initial screen, one line at a time, to the Terminal Handler (A.553, A.502–A.517). When the screen is complete, the SUPERVISOR TASK creates the INPUT TASK, FFT TASK, and OUTPUT TASK (A.554, A.486–A.501). The order of creation is not important for this application, as each task begins by waiting at its input mailbox for frame buffer segments. The SUPERVISOR TASK requests input parameters from the operator (A.556, A.305–A.485). The actual input parameter loop is found at A.478. The loop consists of asking questions (A.479, A.480) until all answers are satisfactory. The operator is asked to choose the highest frequency of interest, number of frames to average, and single runs or continuous runs (A.331–A.353). If the operator answers with an invalid input, the question is repeated (A.365 and A.415). If the operator wishes to abort the questioning by entering a 99, the questions start over from the first question (A.409–A.413). When all three questions have been answered, the operator is asked to confirm his choice (A.482, A.418–A.467). If the operator does not verify the answers, the question number is set to 0 (A.463) and the parameters are requested again. If he confirms the answers as correct, the SUPERVISOR TASK creates up to three frame buffer segments (A.557, A.279–A.304). The SUPERVISOR

TASK places the binary equivalents of the operator answers in the frame segments (A.284, A.292, A.300, A.269–A.278), and sends the segments to the input mailbox for the INPUT TASK (A.287, A.294, A.302).

The SUPERVISOR TASK's background duties are to check its input mailbox for a segment from the OUTPUT TASK and to check a return mailbox for an abort request from the operator (A.558, A.225–A.268). If segments are received at the SUPERVISOR TASK mailbox, the SUPERVISOR TASK sends the segments on to the INPUT TASK if the operator has chosen continuous runs (A.258). Otherwise, the SUPERVISOR TASK deletes the segments (A.242–A.250). When all the segments have been deleted, which halts the FFT analysis, the SUPERVISOR TASK asks for operator input again (A.556).

If an operator abort request is received, the SUPERVISOR TASK, having higher priority than the FFT or OUTPUT TASKS, waits at their mailboxes to intercept the frame segments (A.243, A.249, A.207–A.224). When a segment is received it is deleted (A.219). The SUPERVISOR TASK also checks the INPUT TASK mailbox under abort conditions (A.244). This mailbox is FIFO based to allow the SUPERVISOR TASK to intercept the buffer segment ahead of the higher-priority INPUT TASK (A.523). The SUPERVISOR TASK input mailbox is also checked for frame buffer segments that may have been sent there by the OUTPUT TASK after the abort was requested (A.246). When all of the frame buffer segments have been deleted, the SUPERVISOR TASK asks for operator input (A.556).

INPUT TASK

Listings of the INPUT TASK are in Appendix B. After initializing the buffer for Terminal Handler communications and the mailboxes for communicating with the INTERRUPT TASK and the Terminal Handler (B.335, B.302–B.333), the INPUT TASK waits at its input mailbox for a frame buffer segment (B.338).

When a frame segment is received, the INPUT TASK updates the frame number counter kept by the INPUT TASK (B.340, B.289–B.301), and samples the analog input (B.341, B.231–B.288). The INPUT TASK selects one of two input driver routines, either a software polling loop for faster sampling rates (B.277–B.281), or an INTERRUPT TASK for slower sampling rates (B.251–B.276). If the sampling is driven by interrupts and a Nucleus system call is executing at the time of the interrupt, the time required to respond to that interrupt can vary from 100 to 350 microseconds, depending on the Nucleus call in progress. For the sample rates of 391, 78, and 39 microseconds, corresponding to bandwidths of 1200, 6000, and 12,000 Hz, the system interrupt latency cannot guarantee the precise sampling interval

required. A simple software polling loop with a delay between samples is used for these rates (assembler code for this loop is included after the INPUT TASK listings in Appendix B). This loop operates at priority 0, the highest priority, to guarantee the loop is not interrupted (B.278, B.280) while the sampling is in progress.

For the longer intervals of 3.9 milliseconds and 781 microseconds, corresponding to bandwidths of 120 and 600 Hz, an Interrupt Handler and an INTERRUPT TASK are used (B.251–B.276). Under the iRMX 86 Operating System architecture, an Interrupt Handler is defined as a short procedure with a primary goal of fast interrupt response and limited Nucleus calls. All hardware interrupt levels are masked when an Interrupt Handler is responding to an interrupt. If the interrupt servicing requires higher-level system functions, the Interrupt Handler notifies a waiting INTERRUPT TASK. Higher-level interrupts are enabled when an INTERRUPT TASK is executing. INTERRUPT TASKS can make all system calls.

The INTERRUPT TASK (B.196–B.203) binds the Interrupt Handler to the hardware interrupt level (B.197) and waits for a signal from the Interrupt Handler (B.199). The Interrupt Handler (B.164–B.195) verifies the interval accuracy (B.166–B.173), samples the data (which automatically starts the next sample) (B.175–B.176), places the data in the frame buffer (B.181–B.184), and notifies the INTERRUPT TASK when the frame buffer is full (B.193). If the buffer is not full, the Interrupt Handler resets the interrupt hardware (B.194). The INTERRUPT TASK notifies the INPUT TASK (B.200) and waits for a return message (B.201). The INPUT TASK disables interrupt level 3 (B.274) and returns the token to the INTERRUPT TASK (B.275). The INTERRUPT TASK enables the Interrupt Handler (B.199), but no interrupts will be received from the free-running 8253 Timer because hardware interrupt level 3 has been disabled. Sampling for the next buffer is initiated by simply enabling level 3 (B.272). The INPUT TASK sends a status message to the Terminal Handler (B.342, B.219–B.230) and sends the filled frame buffer to the FFT TASK (B.343). The INPUT TASK then returns to the INPUT TASK input mailbox to wait for the next frame segment (B.338).

FFT TASK

Listings for the FFT TASK are not included with this application note. The FFT TASK is similar in overall format to the INPUT TASK. The FFT TASK waits at its input mailbox for a frame buffer segment from the INPUT TASK. When one is received, the FFT TASK computes the fast Fourier transform of the data. The auto power spectrum is computed and averaged with previous data. The FFT TASK sends its status message to the Terminal Handler for display. If the frame buffer

is the final one to be averaged, the FFT TASK sends the frame buffer to the OUTPUT TASK. If the frame buffer is not the final one in this averaging series, the FFT TASK checks to see if the sampling process is continuous. If so, the frame buffer is returned to the INPUT TASK. If the sampling process is not continuous and the buffer is within two frames of the final frame buffer, the buffer is returned to the SUPERVISOR TASK to prevent unnecessary buffers from going to the INPUT TASK. The FFT TASK then returns to its input mailbox to wait for the next frame buffer.

OUTPUT TASK

Listings for the OUTPUT TASK are not included in this application note. The OUTPUT TASK, like the other tasks, waits at its input mailbox for a buffer. When a frame buffer is received from the FFT TASK, the OUTPUT TASK stores the data in an internal buffer and sends the frame buffer to the SUPERVISOR TASK.

The OUTPUT TASK converts each 32-bit frame buffer value to one of 16 levels by taking the base 2 logarithm of the significant 16 bits of sample value. The display screen is sent to the Terminal Handler one line at a time. Each line of the display is composed of 7 characters of label and y-axis data and 64 characters of display data (reference Figure 11). Each line of the display represents a power of two (from 16 down to 1). The character to be displayed at each location is found by comparing the appropriate sample value against the current line value. If the sample value is greater than the line number, a pound sign is displayed at that location. Otherwise, a space is displayed. The x-axis and labels are sent after the data lines to complete the display. The OUTPUT TASK then waits at its input mailbox for another frame buffer.

TERMINAL HANDLER

The Terminal Handler interfaces to the application tasks via the two mailboxes and buffer segment format shown in Figure 12. If a task wishes to display data, a segment containing the data is sent to the RQ\$TH\$-NORM\$OUT mailbox, specifying a return mailbox. The Terminal Handler displays the data, updates the status fields, and sends the segment to the return mailbox.

Input proceeds in much the same fashion. A task requesting data sends a segment to the RQ\$TH\$NORM\$IN mailbox, again specifying a return mailbox. When the operator terminates the input line with a carriage return, the Terminal Handler puts the data in the segment, updates the status fields, and sends the segment to the return mailbox. This serves two primary purposes: specifying return mailboxes allows multiple tasks to share the display screen while retaining

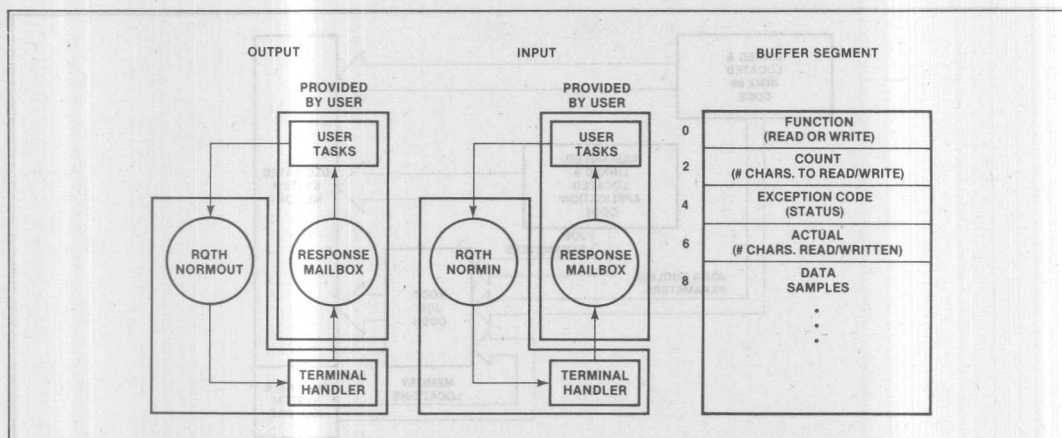


Figure 12. Terminal Handler Interface

synchronization and control over their data buffers; and a user-written Terminal Handler using the same protocol and mailbox names could easily be integrated into the application. For this application, the INPUT TASK, FFT TASK, OUTPUT TASK, and SUPERVISOR TASK all share the screen for output, but only the SUPERVISOR TASK uses the Terminal Handler to obtain operator input.

Nucleus Calls

The iRMX 86 Nucleus provides a comprehensive set of 61 system calls. A complete description of these calls may be found in the iRMX 86 Nucleus Reference Manual. For most applications, only a subset of the 61 calls will be required. The iRMX 86 Nucleus is configurable, which means the final system Nucleus will contain code only for the system calls required for the application. In this case, the following system calls were required:

RQ\$CREATE\$MAILBOX, RQ\$SEND\$MESSAGE, and RQ\$RECEIVE\$MESSAGE provide mailbox management.

RQ\$CREATE\$SEGMENT and RQ\$DELETE\$SEGMENT are used to create and delete segments for the frame buffers, internal buffers, and Terminal Handler.

RQ\$SET\$INTERRUPT, RQ\$EXIT\$INTERRUPT, RQ\$SIGNAL\$INTERRUPT, RQ\$WAIT\$INTERRUPT, RQ\$ENABLE, and RQ\$DISABLE allow the INPUT TASK to handle hardware interrupts knowing only the hardware interrupt level (3).

RQ\$CREATE\$JOB, RQ\$CREATE\$TASK, and RQ\$SET\$PRIORITY are used to create the jobs and tasks, and set the priority of the input polling loop.

RQ\$GET\$TASK\$TOKENS, RQ\$LOOKUP\$OBJECT, RQ\$CATALOG\$OBJECT, RQ\$DISABLE\$DELE-

TION, RQ\$ENABLE\$DELETION, RQ\$GET\$TYPE, RQ\$GET\$PRIORITY, RQ\$GET\$SIZE, and RQ\$SIGNAL\$EXCEPTION are system calls required by the Debugger and the Terminal Handler. None of these calls are necessary in this application if a user-written Terminal Handler is used and debugging is completed.

System Configuration

System Configuration is the integration step in the development process. It consists of selecting the portions of the iRMX 86 Operating System required in the application, mapping this code and the application code to system memory, and creating a Root Job that will initialize the system. The overall configuration process is shown in Figure 13. Configuration requires knowledge of available memory, operating system and application code entry points, priorities, exception handlers, and other system parameters. System Configuration consists of the following steps:

- 1) Selecting the portions of the iRMX 86 Operating System required by the application, including the layers and the specific system calls in each layer.
- 2) Linking and locating those portions.
- 3) Assembling or compiling, linking, and locating the application code.
- 4) Creating a configuration file that will tell the Nucleus the locations of available RAM memory, initial characteristics of each system job, and pertinent overall system parameters. Each job in the system has an entry in the configuration file. The order of the entries is the order of initialization of the jobs.
- 5) Creating the Root Job by assembling, linking, and locating the configuration file.

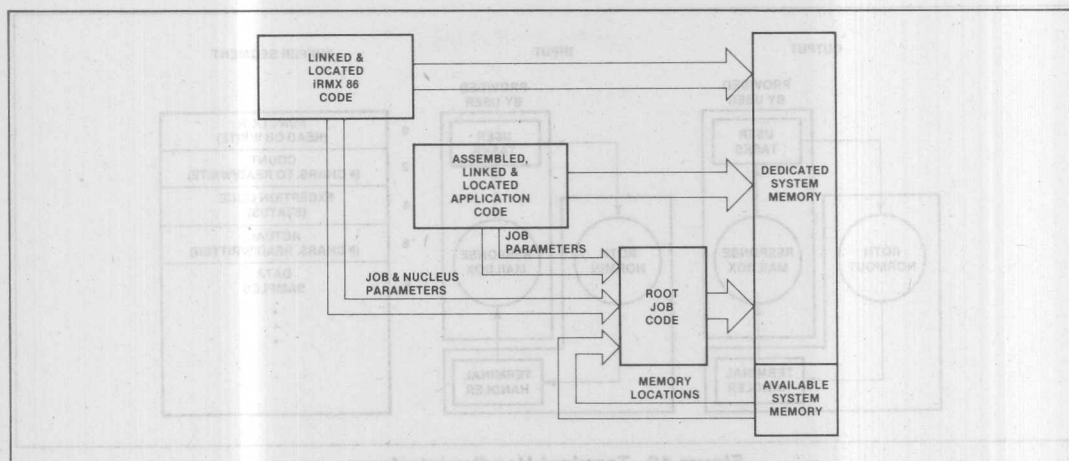


Figure 13. System Configuration Process

During development of an EPROM-based application such as this one, configuration is accomplished twice: once for the RAM-based development system and once for the final EPROM-based system. These configurations are detailed in Appendix C, System Configuration. In both cases, the Root Job that results from configuration initializes the system jobs. For development, the system job structure is shown in Figure 7. The Root Job creates the Debugger Task in the Debugger Job, which in turn creates the Terminal Handler Task. The Root Job then creates the SUPERVISOR TASK, which creates the INPUT TASK, FFT TASK, and OUTPUT TASK. The INPUT TASK creates the INTERRUPT TASK when necessary.

Software development is completed on the iSBC 86/12A Single Board Computer discussed earlier in this note. After application code is debugged and ready to be placed in EPROM memory, the Debugger Job, which contains both the Debugger and the Terminal Handler, is removed and replaced with the Terminal Handler Job, which contains only the Terminal Handler. This job structure is shown in Figure 8. The Nucleus system calls required only by the Debugger are removed from the iRMX 86 Nucleus. The application code is not changed. The application code and the iRMX 86 Nucleus is configured for the final system, put in EPROM memory, and tested on the final hardware system. The final Nucleus and application code required 30.5K bytes of EPROM, allowing room for future code changes and some expansion within the 32K system limit.

The final application hardware is shown in Figure 14. This system contains an iAPX 86/10 CPU, an 8259A Programmable Interrupt Controller, and an 8253 Pro-

grammable Timer. The three chips form the primary hardware requirements for the iRMX 86 Operating System. The system is assembled from Intel components, using standard support circuits and system schematics described in Intel documentation. The analog sampling circuitry is a 12-bit analog to digital converter (ADC) and a sample/hold circuit. Both the sample/hold circuit and the ADC are driven from the on-board local bus. The ADC has a conversion time of 35 microseconds, limiting the overall cycle to approximately 39 microseconds per sample, or a maximum CRT display bandwidth of 12,000 hz.

The hardware system shown in Figure 14 contains components not specifically required for the final configuration. The 8255A Programmable Peripheral Interface and the MULTIBUS multimaster interface are not necessary for a system limited to just spectrum analysis and display via the CRT. However, the flexibility advantages of the iRMX 86 Operating System are supported by this hardware. For instance, the frequency spectrum display is limited by the CRT to a 16-level logarithmic approximation. Accuracy could be improved by using the programmable peripheral interface to drive a plotter or an analog CRT via a digital to analog converter. Software drivers for the plotter or CRT could be new tasks, interfacing to the old tasks through the mailboxes. Or, the OUTPUT TASK could be simply replaced with a new OUTPUT TASK for the plotter and analog CRT. The inclusion of the MULTIBUS interface allows this application to be integrated into a larger system of MULTIBUS-compatible boards. MULTIBUS-compatible memory boards will also aid test and debug. Users of hardware components can include these modular Intel interfaces as required by their application, giving growth and configurability in both hardware and software.

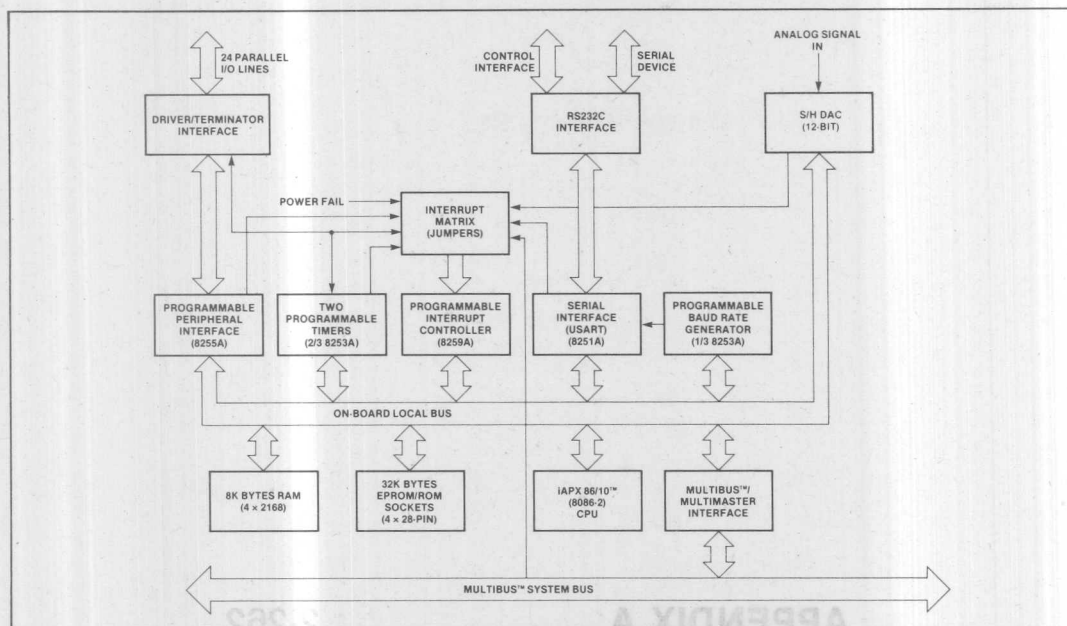


Figure 14. Final Hardware System Block Diagram

SUMMARY

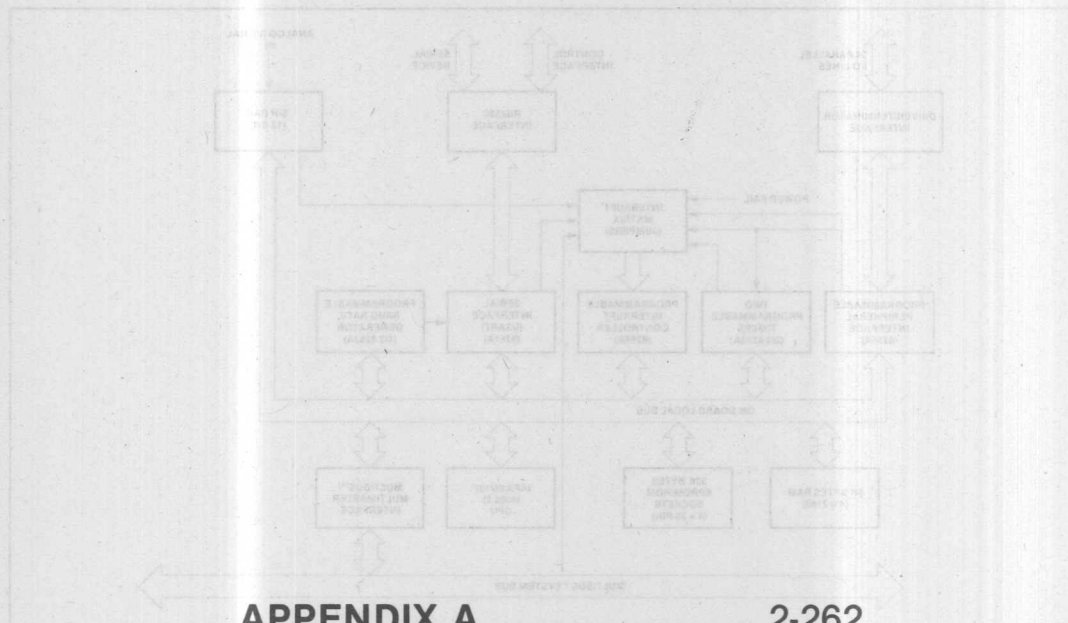
This application note discussed general operating system functions, the Intel iRMX 86 Operating System, using the iRMX 86 Operating System on hardware component systems, and an example of an application implemented in a component environment. Users of the iRMX 86 Operating System are able to simplify application code development through modularity, standard interfaces, freedom from rigid hardware restrictions, and advanced debugging techniques. The iRMX 86 Operating System can be applied to larger systems by adding other iRMX 86 layers, making the software investment beneficial over a wide range of applications.

Operating systems provide many advantages for hardware component designs, but all of these benefits can be utilized only if the operating system and the development environment are fully supported. Intel's support for this application begins with an Intel MDS 230 Series II Microcomputer Development System. The MDS 230 System interfaces with the development hardware through the iSBC 957A™ Monitor. The development hardware is an Intel iSBC 86/12A Single Board Computer, an Intel iSBC 711™ Analog Input Board, an Intel iSBC 032™ 32K RAM Memory Board, an Intel iSBC 064™ 64K RAM Memory Board, and an Intel iSBC 660™ System Chassis. Final application hardware is debugged using Intel's ICE-86™ 8086 In-Circuit Emulator. Software support is provided by the ISIS-II

PL/M 86™ Compiler, MCS-86™ Macro Assembler, and the MCS-86 Utilities LINK86, LOC86, and OH86. The Intel UPP-103™ Universal PROM Programmer is used to convert the final system to PROM memory. This broad support allows expedient development of prototype and final systems based on the iRMX 86 Operating System.

The iRMX 86 Operating Systems and the Intel development tools are valuable only if they translate directly to increased productivity and shortened time to market for new products. This application has 1567 lines of application code. It was developed, from design to final implementation, in approximately 9 man-weeks of effort. This high level of productivity was achieved with the added benefits of modularity, standardization, and ease of application growth.

Intel Corporation is committed to both the continued integration of higher-level functions into hardware and to maintaining compatibility of present software with new hardware. One result of these commitments will be the Intel iAPX 86 and iAPX 286 Processors, which will be compatible with the iRMX 86 Operating System. Another result will be the placement of the iRMX 86 Operating System Nucleus into hardware. This will allow custom hardware applications to have higher-level functions, simplified development, and decreased chip count. Using the iRMX 86 Operating System today will give hardware component users a headstart on Intel's technological innovation for tomorrow.



APPENDIX A	2-262
APPENDIX B	2-283
APPENDIX C	2-300

The Intel iAPX 86 Operating System is designed to convert the final system to PROM memory. This board support allows rapid development of prototype and final system based on the iAPX 86 Operating System.

The iAPX 86 Operating System and the final development tools are valuable only if they translate directly to increased productivity and shortened time to market for new products. This application has 1507 lines of application code. It was developed from design to final implementation, in approximately 2 man-weeks of effort. This high level of productivity was achieved with the added benefits of modularity, standardization, and ease of application growth.

Intel Corporation is committed to both the continued integration of higher-level functions into hardware and maintaining compatibility of present software with new hardware. One result of these commitments will be the Intel iAPX 86 and iAPX 286 processors, which will be compatible with the iAPX 86 Operating System. Another result will be the presence of the iAPX 86 Operating System Nucleus into hardware. This will allow custom hardware applications to have higher-level functions, simplified development, and decreased chip count. Using the iAPX 86 Operating System today will give hardware component users a headstart on Intel's technological innovation for tomorrow.

The application note discusses general operating system functions, the Intel iAPX 86 Operating System, using the iAPX 86 Operating System on hardware components, and an example of an application implemented in a component environment. Users of the iAPX 86 Operating System are able to simplify application code development through modularity, standardization, freedom from rigid hardware restrictions, and advanced debugging techniques. The iAPX 86 Operating System can be applied to larger systems by adding other iAPX 86 bytes, making the software environment beneficial over a wide range of applications.

Operating systems provide many advantages for hardware component designs. But all of these benefits can be utilized only if the operating system and the development environment are fully supported. Intel's support for the application begins with an Intel MDS 230 Series II Microcomputer Development System. The MDS 230 System interfaces with the development hardware through the iAPX 86 Monitor. The development hardware is an Intel iAPX 86 Single Board Computer, an Intel iAPX 86 Analog Input Board, an Intel iAPX 86 RAM Memory Board, an Intel iAPX 86 RAM Memory Board, and an Intel iAPX 86 System Channel. Final application hardware is developed using Intel's iAPX 86 In-Circuit Emulator. Software support is provided by the iAPX 86

PL/M-86 COMPILER SUPERVISOR TASK FOR AP NOTE 110, OCTOBER 1980
 ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE SUPERVISOR_MODULE
 OBJECT MODULE PLACED IN :F1:superv.OBJ
 COMPILER INVOKED BY: plm86 :F1:superv.p86

```

      stitle(' SUPERVISOR TASK FOR AP NOTE 110, OCTOBER 1980')
      slarge debug
      SUPERVISOR_MODULE:
      do;
      $include(:f1:nuclus.ext)
      = $SAVE NOLIST

089 1 declare token      literally 'word';

      /******
      /* The six mailboxes immediately following will */
      /* form all of the inter-task communications */
      /* interfaces for the five tasks that run in */
      /* this system. */
      /******

090 1 declare th_in_mbx      token;
091 1 declare th_out_mbx     token;
092 1 declare to_in_mbx      token public;
093 1 declare to_fft_mbx     token public;
094 1 declare to_out_mbx     token public;
095 1 declare to_supervisor_mbx token public;

096 1 declare return_th_in_mbx token;
097 1 declare return_th_out_mbx token;
098 1 declare dummy_mbx      token;
099 1 declare frame_segment_one token;
100 1 declare frame_segment_two token;
101 1 declare frame_segment_three token;
102 1 declare th_in_segment_token token;
103 1 declare th_out_segment_token token;

104 1 declare general_index      byte;
105 1 declare number_of_fft_data_segments byte;

106 1 declare insert_text_pointer pointer;

107 1 declare abort_flag      word;
108 1 declare status          word;
109 1 declare segment_deleted_tally word;
110 1 declare text_length      word;

111 1 declare root_job_token    token;
112 1 declare input_task_token token;
113 1 declare fft_task_token    token;
114 1 declare output_task_token token;

115 1 declare parameters      structure(
      actual_samples      word,

```



```

actual_interval      word,
frequency_answer(5)  byte,
actual_frames_to_average word,
frames_to_average_answer(5) byte,
continuous_flag      word,
continuous_flag_answer(5) byte);

116 1 declare abort      literally 'OFFH';
117 1 declare ascii_9    literally '039H';
118 1 declare carriage_return literally '0DH';
119 1 declare forever    literally 'while 1';
120 1 declare line_feed  literally '0AH';
121 1 declare new_fft_run literally 'OFFH';
122 1 declare no_abort    literally '00H';
123 1 declare no_response_requested literally '00H';
124 1 declare null        literally '00H';
125 1 declare queue_fifo  literally '00H';
126 1 declare queue_priority literally '01H';
127 1 declare rootjob     literally '03H';
128 1 declare run_continuous literally 'OFFFH';
129 1 declare size_120_bytes literally '120';
130 1 declare space       literally '20H';
131 1 declare supervisor_job literally '00H';
132 1 declare th_read      literally '01H';
133 1 declare th_write     literally '05H';
134 1 declare wait_forever literally 'OFFFH';

/* The following declaration sets the characters */
/* to send the cursor home at the beginning of */
/* each message to the display screen. The */
/* sequence is tilde, DC2 (Hazeltine)). */

135 1 declare cursor_home_chars(2) byte data (07EH,012H);

136 1 declare frame_pointer      pointer;
137 1 declare frame_pointer_values structure(
    offset word,
    base word) at (@frame_pointer);

138 1 declare frame based frame_pointer structure(
    samples_per_frame word,
    sample_interval word,
    frames_to_average word,
    continuous_flag word,
    this_frame_number word,
    number_samples_missed word,
    sample_pointer word,
    reset_flag word,
    sample(256) integer);

139 1 declare th_in_segment_pointer      pointer;
140 1 declare th_in_segment_pointer_values structure(
    offset word,
    base word) at (@th_in_segment_pointer);

```

```

141 1 declare th_in_segment based th_in_segment_pointer
      structure(
        function word,
        count word,
        exception_code word,
        actual word,
        message(112) byte);

142 1 declare th_out_segment_pointer pointer;
143 1 declare th_out_segment_pointer_values structure(
      offset word,
      base word) at (@th_out_segment_pointer);

144 1 declare th_out_segment based th_out_segment_pointer
      structure(
        function word,
        count word,
        exception_code word,
        actual word,
        home_chars(2) byte,
        line_index(24) byte,
        message(84) byte);

145 1 declare frequency_question(*) byte data ('Please'
      ' enter the highest frequency in Hz (120,
      ' 600, 1200, 6000, OR 12000):');

146 1 declare frequency_answers data(*) byte data (05H,
      03H,03H,04H,04H,05H,0H,
      ' 120 600 1200 600012000 ',
      42H,0FH, 00DH,03H, 087H,01H,
      4EH,00H, 027H,00H, 00H,000H);

147 1 declare average_question(*) byte data ('Please'
      ' enter the number of frames to average'
      ' (1, 2, 4, 8, 16, OR 32):');

148 1 declare average_answers data(*) byte data (06H,
      01H,01H,01H,01H,02H,02H,
      ' 1 2 4 8 16 32',
      01H,00H, 02H,00H, 04H,00H,
      08H,00H, 10H,00H, 20H,00H);

149 1 declare continuous_question(*) byte data ('Please'
      ' enter 'I' for one sample run or 'C'
      ' for continuous running:');

150 1 declare continuous_answers data(*) byte data (03H,
      01H,01H,01H,00H,00H,00H,
      ' 1 C C ',
      00H,00H, 0FFH,0FFH, 0FFH,0FFH,
      00H,00H, 00H,00H, 00H,00H);

151 1 declare reject_message(*) byte data ('I cannot'
      ' accept your answer. PLEASE TRY AGAIN.');
```

```

152 1 declare status_line_one(*) byte data ('Current'
      ' settings are the following: frequency'
      ' range 0 to Hz,');

153 1 declare status_line_two(*) byte data ('
      ' frames to average per output display,'
      ' and ');

154 1 declare continuous_runs(*) byte data
      ('continuous runs. ');

155 1 declare single_run(*) byte data
      ('a single run. ');

156 1 declare go_ahead_question(*) byte data ('If '
      ' these settings are correct, enter 'G' '
      ' to begin running. ');

157 1 declare header_line_one(*) byte data (' INTEL'
      ' APNOTE 110--THE iRMX 86 OPERATING SYSTEM'
      ' AND iAPX 86 COMPONENT DESIGNS. ');

      /*****
      /* The following three procedure declarations */
      /* link the tasks outside the SUPERVISOR MODULE */
      /* with the supervisor task. */
      *****/

158 1 INPUT_TASK: PROCEDURE EXTERNAL;
159 2 END INPUT_TASK;

160 1 FFT_TASK: PROCEDURE EXTERNAL;
161 2 END FFT_TASK;

162 1 OUTPUT_TASK: PROCEDURE EXTERNAL;
163 2 END OUTPUT_TASK;

      /*****
      *****/
      /** The following five procedures are general **/
      /** utility procedures called by the primary **/
      /** working procedures. **/
      *****/
      /*****
      /* Blank line just fills the message buffer */
      /* with Blank characters. */
      *****/

164 1 BLANK_LINE: PROCEDURE;
165 2 declare blank_line_index word;

```

```

166 2      do blank_line_index = 0 to 78;
167 3          th_out_segment.message (blank_line_index)
168 3              = space;
169 2      th_out_segment.message (79) = carriage_return;
170 2      END BLANK_LINE;

/*****
/* DISPLAY_LINE sends the message to the */
/* terminal handler output mailbox and */
/* waits for the segment to be returned. */
*****/

171 1      DISPLAY_LINE: PROCEDURE;
172 2      call rq$send$message (th_out_mbx,
173 2          th_out_segment.token,
174 2          return_th_out_mbx, @status);
175 2      th_out_segment.token = rq$receive$message
176 2          (return_th_out_mbx,
177 2          wait_forever, @dummy_mbx,
178 2          @status);
179 2      END DISPLAY_LINE;

/*****
/* INSERT_TEXT fills the output message segment */
/* with the chosen message and pads the rest of */
/* the line with blanks. */
*****/

180 1      INSERT_TEXT: PROCEDURE(TEXT_POINTER, HOW_MANY);
181 2      declare text_pointer      pointer;
182 2      declare how_many          word;
183 2      declare dummy_based_text_pointer structure(
184 2          entries(80) byte);
185 2      declare insert_text_index word;
186 2      do insert_text_index = 0 to (how_many - 1);
187 3          th_out_segment.message (insert_text_index) =
188 3              dummy_based_text_pointer.entries (insert_text_index);
189 3      end;
190 2      do while insert_text_index < 79;
191 3          th_out_segment.message (insert_text_index)
192 3              = space;
193 3          insert_text_index = insert_text_index + 1;
194 3      end;
195 2      th_out_segment.message (79) = carriage_return;
196 2      END INSERT_TEXT;

```



```

/*****
/* Move_down_line puts the required number of */
/* line_feed_characters in the first 6th      */
/* through 29th character of the output       */
/* message. Each line is displayed on the     */
/* screen in its proper location. This allows */
/* multiple tasks to access the screen       */
/* without having to blank the line each     */
/* time. This technique assumes each message */
/* sends the cursor home each time.          */
*****/

189 1  MOVE_DOWN_LINE: PROCEDURE(SKIP_LINES);
190 2  declare skip_lines          word;
191 2  declare move_down_line_index word;

192 2  if skip_lines > 0 then
193 2      do move_down_line_index = 0 to (skip_lines - 1);
194 3      th_out segment.line_index (move_down_line_index)
        = line_feed;
195 3  end;

196 2  do move_down_line_index = skip_lines to 23;
197 3  th_out segment.line_index (move_down_line_index)
        = null;
198 3  end;

199 2  END MOVE_DOWN_LINE;

/*****
/* SEND_REJECT_MESSAGE is called by INPUT      */
/* PARAMETERS. It just sends a reject message */
/* to the CRT to inform the user that the     */
/* answer the user gave was not valid.        */
*****/

200 1  SEND_REJECT_MESSAGE: PROCEDURE;
201 2  call move_down_line (21);
202 2  text_length          = size(reject_message);
203 2  insert_text_pointer  = @reject_message;
204 2  call insert_text (insert_text_pointer, text_length);
205 2  call display_line;

206 2  END SEND_REJECT_MESSAGE;

/*****
/*****
/** The following procedures are the primary **/
/** working procedures called by the supervisor **/
/** procedure and its called procedures.    **/
/*****
/*****

```

```

/*****
/* PURGE MAILBOX removes all tokens from
/* a mailbox. The purpose is to remove segments
/* waiting for processing by one of the tasks
/* if the operator has specified an abort
/* request. If the segment deletion was
/* successful, PURGE MAILBOX updates the
/* segment deleted tally.
*****/

207 1  PURGE_MAILBOX: PROCEDURE(MAILBOX_TO_PURGE);
208 2  declare mailbox_to_purge token;
209 2  declare for_110_milliseconds literally 'OBH';
210 2  declare message_received literally 'OH';
211 2  declare contents_token token;
212 2  declare purge_dummy_mbx token;
213 2  declare purge_status token;
214 2  purge_status = message_received;
215 2  do while purge_status = message_received;
216 3  contents_token = rq$receive$message
      (mailbox_to_purge, for_110_milliseconds,
      @purge_dummy_mbx, @purge_status);
217 3  if purge_status = message_received then
218 3  do;
219 4  call rq$delete$segment
      (contents_token, @purge_status);
220 4  segment_deleted_tally =
      segment_deleted_tally + 1;
221 4  purge_status = message_received;
222 4  end;
223 3  end;
224 2  END PURGE_MAILBOX;

/*****
/* MONITOR MAILBOXES polls the
/* return_th in mailbox and the
/* to_supervisor_mailbox for messages. The
/* messages will be abort (from the operator),
/* and FFT done or OUTPUT done if the runs are
/* not continuous. If the runs are not
/* continuous and an FFT done message is
/* received, MONITOR_MAILBOXES will initialize
/* the OUTPUT task.
*****/

225 1  MONITOR_MAILBOXES: PROCEDURE;

```

```

226 2      declare cannot_wait      literally '00H';
227 2      declare done              literally '0FFH';
228 2      declare for_400_milliseconds literally '28H';
229 2      declare message_received  literally '00H';
230 2      declare not_done          literally '00H';

231 2      declare monitor_dummy_mbx token;
232 2      declare monitor_token     token;
233 2      declare monitor_status    token;

234 2      declare done_flag         byte;
235 2      declare monitor_index     word;

236 2      done_flag = not_done;

237 2      segment_deleted_tally = 0;
238 2      call rq$send$message
        (th_in_mbx, th_in_segment_token,
         return_th_in_mbx, @status);

239 2      do while done_flag = not_done;
240 3          /* Check for operator input here. */
        monitor_token = rq$receive$message
        (return_th_in_mbx, for_400_milliseconds,
         @monitor_dummy_mbx, @monitor_status);
241 3      if monitor_status = message_received then
242 3          do while segment_deleted_tally <
        number_of_fft_data_segments;
243 4              call purge_mailbox (to_fft_mbx);
244 4              if segment_deleted_tally <
        number_of_fft_data_segments then
245 4                  call purge_mailbox (to_input_mbx);
246 4              if segment_deleted_tally <
        number_of_fft_data_segments then
247 4                  call purge_mailbox (to_supervisor_mbx);
248 4              if segment_deleted_tally <
        number_of_fft_data_segments then
249 4                  call purge_mailbox (to_output_mbx);
250 4              done_flag = done;
251 4          end;

252 3      if done_flag = not_done then
253 3          do;
254 4              monitor_token = rq$receive$message
        (to_supervisor_mbx, for_400_milliseconds,
         @monitor_dummy_mbx, @monitor_status);
255 4              if monitor_status = message_received then
256 4                  do;
257 5                      if parameters.continuous_flag =
        run_continuous then
258 5                          call rq$send$message
        (to_input_mbx, monitor_token,
         no_response_requested,
         @monitor_status);
                else

```

```

259 5      do;
260 6      call rq$delete$segment
          (monitor_token, @monitor_status);
261 6      segment_deleted_tally
          = segment_deleted_tally + 1;
262 6      if segment_deleted_tally
          = number_of_fft_data_segments
          then done_flag = done;
264 6      end;
265 5      end;
266 4      end;
267 3      end;
268 2      END MONITOR_MAILBOXES;

/*****
/* SET_SEGMENT initializes the common parameter */
/* areas of the segment. The pointer to the */
/* proper segment is set up by */
/* INITIALIZE_SEGMENTS. */
*****/

269 1      SET_SEGMENT: PROCEDURE;
270 2      frame.samples_per_frame = 128;
271 2      frame.sample_interval
          = parameters.actual_interval;
272 2      frame.frames_to_average
          = parameters.actual_frames_to_average;
273 2      frame.continuous_flag = parameters.continuous_flag;
274 2      frame.this_frame_number = 00H;
275 2      frame.number_samples_missed = 00H;
276 2      frame.sample_pointer = 00H;
277 2      frame.reset_flag = 00H;
278 2      END SET_SEGMENT;

/*****
/* INITIALIZE_SEGMENTS creates the three FFT */
/* data segments and calls SET_SEGMENT for each */
/* segment to initialize the common parameter */
/* areas of the segments. */
*****/

279 1      INITIALIZE_SEGMENTS: PROCEDURE;
280 2      declare size_528_bytes literally '528';
281 2      frame_pointer_values.offset = 0;
282 2      frame_segment_one = rq$create$segment
          (size_528_bytes, @status);
283 2      frame_pointer_values.base = frame_segment_one;
284 2      call set_segment;

```



```

285 2      frame.reset_flag      = new_fft_run;
286 2      number_of_fft_data_segments = 1;
287 2      call rq$send$message
          (to_input_mbx, frame_segment one,
           no_response_requested, @status);
288 2      if parameters.actual_frames_to_average > 1 then
289 2          do;
290 3              frame_segment_two = rq$create$segment
                                   (size 528_bytes, @status);
291 3              frame_pointer_values.base = frame_segment_two;
292 3              call set_segment;
293 3              number_of_fft_data_segments = 2;
294 3              call rq$send$message
                                   (to_input_mbx, frame_segment two,
                                   no_response_requested, @status);
295 3          end;
296 2      if parameters.actual_frames_to_average > 2 then
297 2          do;
298 3              frame_segment_three = rq$create$segment
                                   (size 528_bytes, @status);
299 3              frame_pointer_values.base
                                   = frame_segment_three;
300 3              call set_segment;
301 3              number_of_fft_data_segments = 3;
302 3              call rq$send$message
                                   (to_input_mbx, frame_segment three,
                                   no_response_requested, @status);
303 3          end;
304 2      END INITIALIZE_SEGMENTS;

          /******
          /* INPUT_PARAMETERS contains three procedures:
          /* set_question_pointers, get_answer,
          /* and verify_answers. The INPUT_PARAMETERS
          /* loop consists of calls to these three
          /* procedures and, as usual, exists at the end
          /* of the procedure.
          /******

305 1      INPUT_PARAMETERS: PROCEDURE;
306 2      declare actual_pointer      pointer;
307 2      declare answer_pointer      pointer;
308 2      declare answer_display_pointer pointer;
309 2      declare question_pointer    pointer;

310 2      declare answer_actual_value based
          actual_pointer word;
311 2      declare answer_overlay based
          answer_pointer structure(
          number_of_answers byte,

```

```

length_of_answer(6) byte,
values_to_match(30) byte,
really_are(6) word;

312 2 declare answer_display based
      answer_display_pointer structure(
      characters(5) byte);

313 2 declare answer_byte_index byte;
314 2 declare answer_index byte;
315 2 declare answer_match byte;
316 2 declare byte_match byte;
317 2 declare input_byte_index byte;
318 2 declare output_byte_index byte;
319 2 declare question_number byte;
320 2 declare stop_byte byte;

321 2 declare ascii_small_g literally '067H';
322 2 declare ascii_capital_G literally '047H';
323 2 declare average_entry_point literally '0';
324 2 declare continuous_entry_point literally '48';
325 2 declare frequency_entry_point literally '58';
326 2 declare match literally '0FFH';
327 2 declare no_match literally '00H';
328 2 declare not_negative literally '< 255';
329 2 declare nothing_returned literally '00H';

330 2 set_question_pointers: procedure;
331 3 do case question_number;
332 4 do;
333 5 text_length = size (frequency_question);
334 5 question_pointer = @frequency_question;
335 5 answer_pointer = @frequency_answers_data;
336 5 actual_pointer = @parameters.actual_interval;
337 5 answer_display_pointer
      = @parameters.frequency_answer;
338 5 end;

339 4 do;
340 5 text_length = size (average_question);
341 5 question_pointer = @average_question;
342 5 answer_pointer = @average_answers_data;
343 5 actual_pointer
      = @parameters.actual_frames_to_average;
344 5 answer_display_pointer
      = @parameters.frames_to_average_answer;
345 5 end;

346 4 do;
347 5 text_length = size (continuous_question);
348 5 question_pointer = @continuous_question;
349 5 answer_pointer = @continuous_answers_data;
350 5 actual_pointer = @parameters.continuous_flag;

```

```

351 5      answer_display_pointer
352 5      = @parameters.continuous_flag_answer;
353 4      end;
354 3      end set_question_pointers;

355 2      get_answer: procedure;
          /* First display the question to be answered */
          /* by the operator. */
356 3      call insert_text (question_pointer, text_length);
357 3      call move_down_line (19);
358 3      call display_line;

          /* Then blank the line below for an answer line. */
359 3      call blank_line;
360 3      call move_down_line (20);
361 3      call display_line;

          /* Now wait for a response from the operator. */
362 3      call rq$send$message
          (th_in_mbx, th_in_segment_token,
           return_th_in_mbx, @status);
363 3      th_in_segment_token = rq$receive$message
          (return_th_in_mbx, wait_forever,
           @dummy_mbx, @status);
364 3      th_in_segment_pointer values.base
          = th_in_segment_token;

          /* If there is no message returned then send */
          /* a reject message. */
365 3      if th_in_segment.actual = nothing_returned
          then call send_reject_message;

          /* Otherwise it is time to check the response */
          /* against the possible answers. */
          else
367 3      do;
368 4      answer_match = no_match;

          /* Set the number of possible answers. */
369 4      answer_index
          = answer_overlay.number_of_answers;

          /* Start a loop to check all of the */
          /* possible answers. */

```

```

370 4 do while (answer_match = no_match) and
      (answer_index > 0);
      /* Set the starting point for the */
      /* byte by byte compare.          */
371 5 answer_byte_index = (answer_index * 5) - 1;
      /* Set the stopping point for */
      /* the compare.                */
372 5 stop_byte = answer_byte_index
      - answer_overlay.length_of_answer
      (answer_index - 1);
      /* Start with a "match" so we can */
      /* check until "no match" occurs. */
373 5 byte_match = match;
      /* Set starting point at the right end */
      /* of the input data (allows us to    */
      /* ignore leading blanks and the     */
      /* ending carriage return).          */
374 5 input_byte_index = th_in_segment.actual-2;
      /* Scan the bytes until all pertinent */
      /* ones are checked or a "no_match"  */
      /* occurs.                            */
375 5 do while (byte_match = match) and
      (answer_byte_index > stop_byte);
376 6 if (input_byte_index not_negative)
      then do;
378 7 if th_in_segment.message
      (input_byte_index) =
      answer_overlay.values_to_match
      (answer_byte_index)
      then byte_match = match;
380 7 else byte_match = no_match;
381 7 end;
382 6 else byte_match = no_match;
383 6 answer_byte_index = answer_byte_index-1;
384 6 input_byte_index = input_byte_index-1;
385 6 end;
      /* A "match" at this point means ALL */
      /* bytes matched.                      */
386 5 if byte_match = match then
387 5 do;
      /* Set real values via          */

```



```

/* answer_actual_value overlay. */
388 6 answer_actual_value
      = answer_overlay.really_are
      (answer_index - 1);
389 6 answer_match = match;
      /* Insert displayable values for */
      /* later display. */
390 6 answer_byte_index = 4;
391 6 input_byte_index = (answer_index*5)-1;
392 6 do while answer_byte_index not negative;
393 7   answer_display.characters
      (answer_byte_index)
      = answer_overlay.values_to_match
      (input_byte_index);
394 7   input_byte_index
      = input_byte_index - 1;
395 7   answer_byte_index
      = answer_byte_index - 1;
396 7 end;
      /* We got a match, so be sure the */
      /* reject message line is blanked. */
397 6 call move_down_line (21);
398 6 call blank_line;
399 6 call display_line;
400 6 end;
      /* If no match, then let's compare the */
      /* input with the next possible answer. */
/ 401 5 else answer_index = answer_index - 1;
402 5 end;
      /* If we got a match, then we can move on */
      /* to the next question. */
403 4 if answer_match = match
      then question_number = question_number + 1;
      /* Otherwise we have to check for an */
      /* abort request of '99'. */
      else
405 4   do;
406 5   input_byte_index = th_in_segment.actual-2;
407 5   if (th_in_segment.message(input_byte_index)
      = ascii_9)
      and
      (th_in_segment.message (input_byte_index -
1)
      = ascii_9) then

```

```

/* Abort requests are valid, so blank */
/* the reject message line and reset */
/* the question number so we start */
/* asking all over. */
408 5 do;
409 6 question_number = 1;
410 6 call move down line (21);
411 6 call blank_line;
412 6 call display_line;
413 6 end;
else
/* But if nothing matched and the */
/* answer was not an abort request, */
/* then we have to ask the operator */
/* to try again on this question. */
414 5 call send_reject_message;
415 5 end;
416 4 end;

417 3 end get_answer;

418 2 verify_answers: procedure;

/* First put the output line in the buffer. */
419 3 text_length = size (status_line_one);
420 3 call insert_text (@status_line_one, text_length);

/* Then insert the displayable frequency answer. */
421 3 input_byte_index = 0;
422 3 stop_byte = frequency_entry_point + 4;
423 3 do output_byte_index
= frequency_entry_point to stop_byte;
424 4 th_out_segment.message (output_byte_index) =
parameters.frequency_answer (input_byte_index);
425 4 input_byte_index = input_byte_index + 1;
426 4 end;

427 3 call move down line(19);
428 3 call display_line;

/* We have sent the first line, now it is time */
/* to get the second line. */
429 3 text_length = size (status_line_two);
430 3 call insert_text (@status_line_two, text_length);

/* We have to insert the displayable "frames */

```

```

/* to average" answer. */
431 3 input_byte_index = 0;
432 3 stop_byte = average_entry_point + 4;
433 3 do output_byte_index
    = average_entry_point to stop_byte;
434 4 th_out_segment.message (output_byte_index) =
    parameters.frames_to_average_answer
    (input_byte_index);
435 4 input_byte_index = input_byte_index + 1;
436 4 end;

/* The continuous answer is different--we have */
/* to decide if we have continuous runs or */
/* single runs, and insert those words in the */
/* display line. */

437 3 input_byte_index = 0;
438 3 stop_byte = continuous_entry_point + 15;
439 3 if parameters.continuous_flag = run_continuous
    then
440 3 do output_byte_index
    = continuous_entry_point to stop_byte;
441 4 th_out_segment.message (output_byte_index) =
    continuous_runs (input_byte_index);
442 4 input_byte_index = input_byte_index + 1;
443 4 end;
    else
444 3 do output_byte_index
    = continuous_entry_point to stop_byte;
445 4 th_out_segment.message (output_byte_index) =
    single_run (input_byte_index);
446 4 input_byte_index = input_byte_index + 1;
447 4 end;

/* Then send the message and wait for a response */
/* from the operator. */

448 3 call move_down_line(20);
449 3 call display_line;

450 3 text_length = size (go_ahead_question);
451 3 call insert_text (@go_ahead_question, text_length);
452 3 call move_down_line (21);
453 3 call display_line;

454 3 call blank_line;
455 3 call move_down_line(22);
456 3 call display_line;

457 3 call rq$send$message
    (th_in_mbx, th_in_segment_token,
    return_th_in_mbx, @status);
458 3 th_in_segment_token
    = rq$receive$message

```

```

                                (return_th_in_mbx, wait_forever,
                                @dummy_mbx, @status);
459   3   th_in_segment_pointer_values.base
                                = th_in_segment.token;
460   3   input_byte_index = th_in_segment.actual - 2;

                                /* Check for a "g" or "G" (we aren't fussy). If */
                                /* we got it, let's quit asking the selection */
                                /* questions and go. If not, we have to start */
                                /* at question 1 again rather than try to find */
                                /* out which of his or her answers wasn't */
                                /* acceptable. */
                                /*
461   3   if (th_in_segment.message (input_byte_index)
                                = ascii_small_g) or
                                (th_in_segment.message (input_byte_index)
                                = ascii_capital_G) then;
463   3   else question_number = 0;

464   3   call blank_line;
465   3   call move_down_line (21);
466   3   call display_line;

467   3   end verify_answers;

                                /* * * * * * */
                                /* As usual, the actual INPUT PARAMETERS control */
                                /* loop is at the end. */
                                /* * * * * * */

468   2   question_number = 0;

                                /* All we do is get the next question, ask the */
                                /* question until it is answered successfully, */
                                /* ask all of the questions, then check all of */
                                /* the answers. If the operator doesn't like */
                                /* the set of answers, we loop through them */
                                /* again. First we make sure the reject message */
                                /* line and other pertinent lines start out */
                                /* blanked. */

469   2   call blank_line;
470   2   call move_down_line (18);
471   2   call display_line;
472   2   call move_down_line (21);
473   2   call display_line;
474   2   call move_down_line (22);
475   2   call display_line;
476   2   call move_down_line (23);
477   2   call display_line;

478   2   input_loop: do while question_number < 3;
479   3   call set_question_pointers;
480   3   call get_answer;

```



```

481 3      end;
482 2      call verify_answers;
483 2      if question_number = 0 then goto input_loop;
485 2      END INPUT_PARAMETERS;

/* ***** */
/* INITIALIZE TASKS initializes the INPUT TASK */
/* and the FFT TASK. If the FFT runs are to be */
/* continuous, INITIALIZE TASK also initializes */
/* the OUTPUT TASK. If the runs are not */
/* continuous, the OUTPUT TASK is initialized */
/* by MONITOR MAILBOXES. */
/* ***** */

486 1      INITIALIZE_TASKS: PROCEDURE;

487 2      declare hardware_interrupt_level_3 literally '038H';
488 2      declare no_data_segment literally '00H';
489 2      declare nucleus_allocated_stack literally '00H';
490 2      declare software_priority_level_67 literally '67';
491 2      declare software_priority_level_130 literally '130';
492 2      declare software_priority_level_131 literally '131';
493 2      declare stack_size_512 literally '512';
494 2      declare task_flags literally '00H';

495 2      input_task_token = rq$create$task
      (software_priority_level_67, @input_task,
      no_data_segment, nucleus_allocated_stack,
      stack_size_512, task_flags, @status);

496 2      call rq$catalog$object
      (supervisor_job, input_task_token,
      @(10, 'INPUT TASK'), @status);

497 2      fft_task_token = rq$create$task
      (software_priority_level_131, @fft_task,
      no_data_segment, nucleus_allocated_stack,
      stack_size_512, task_flags, @status);

498 2      call rq$catalog$object
      (supervisor_job, fft_task_token,
      @(8, 'FFT TASK'), @status);

499 2      output_task_token = rq$create$task
      (software_priority_level_130, @output_task,
      no_data_segment, nucleus_allocated_stack,
      stack_size_512, task_flags, @status);

500 2      call rq$catalog$object
      (supervisor_job, output_task_token,
      @(11, 'OUTPUT TASK'), @status);

```

```

501 2      END INITIALIZE_TASKS;
          /******
          /* Initial screen displays the initial two
          /* lines on the screen and sends blank lines
          /* for all the other lines of the first screen.
          /******
502 1      INITIAL_SCREEN: PROCEDURE;
503 2      declare initial_screen_index word;
504 2      call move_down_line(0);
505 2      call blank_line;
506 2      call display_line;
507 2      call move_down_line(1);
508 2      text_length = size(header_line one);
509 2      insert_text_pointer = @header_line one;
510 2      call insert_text(insert_text_pointer, text_length);
511 2      call display_line;
512 2      call blank_line;
513 2      do initial_screen_index = 2 to 23;
514 3          call move_down_line(initial_screen_index);
515 3          call display_line;
516 3      end;
517 2      END INITIAL_SCREEN;
          /******
          /* INITIALIZE_BUFFERS takes care of the
          /* initialization required for general
          /* SUPERVISOR TASK start up.
          /******
518 1      INITIALIZE_BUFFERS: PROCEDURE;
519 2      return_th_in_mbx = rq$create$mailbox
          (queue_fifo, @status);
520 2      call rq$catalog$object
          (supervisor_job, return_th_in_mbx,
          @('9','SUP TH IN'), @status);
521 2      return_th_out_mbx = rq$create$mailbox
          (queue_fifo, @status);
522 2      call rq$catalog$object
          (supervisor_job, return_th_out_mbx,
          @('10','SUP TH OUT'), @status);
523 2      to_input_mbx = rq$create$mailbox
          (queue_fifo, @status);
524 2      call rq$catalog$object
          (supervisor_job, to_input_mbx,
          @('12','TO INPUT MBX'), @status);
525 2      to_fft_mbx = rq$create$mailbox
          (queue_priority, @status);

```

```

526 2      call rq$catalog$object
              (supervisor_job, to_fft_mbx,
              @(10,'TO FFT MBX'), @status);
527 2      to_output_mbx = rq$create$mailbox
              (queue_priority, @status);
528 2      call rq$catalog$object
              (supervisor_job, to_output_mbx,
              @(10,'TO OUT MBX'), @status);
529 2      to_supervisor_mbx = rq$create$mailbox
              (queue_priority, @status);
530 2      call rq$catalog$object
              (supervisor_job, to_supervisor_mbx,
              @(10,'TO SUP MBX'), @status);

531 2      root_job_token = rq$get$task$tokens
              (root_job, @status);
532 2      th_out_mbx = rq$lookup$object
              (root_job_token, @(11,'RQTHNORMOUT'),
              wait_forever, @status);
533 2      th_in_mbx = rq$lookup$object
              (root_job_token, @(10,'RQTHNORMIN'),
              wait_forever, @status);

534 2      th_in_segment_token = rq$create$segment
              (size_120_bytes, @status);
535 2      call rq$catalog$object
              (supervisor_job, th_in_segment_token,
              @(10,'S THIN SEG'), @status);
536 2      th_in_segment_pointer_values.offset = 0;
537 2      th_in_segment_pointer_values.base
              = th_in_segment_token;
538 2      th_in_segment.function = th_read;
539 2      th_in_segment.count = 82;

540 2      th_out_segment_token = rq$create$segment
              (size_120_bytes, @status);
541 2      call rq$catalog$object
              (supervisor_job, th_out_segment_token,
              @(11,'S THOUT SEG'), @status);
542 2      th_out_segment_pointer_values.offset = 0;
543 2      th_out_segment_pointer_values.base
              = th_out_segment_token;
544 2      th_out_segment.function = th_write;
545 2      th_out_segment.count = 111;

546 2      do general_index = 0 to 2;
547 3      th_out_segment.home_chars (general_index)
              = cursor_home_chars (general_index);
548 3      end;
549 2      END INITIALIZE_BUFFERS;

/*****
/* At last, the SUPERVISOR TASK! All it does is */

```

```

/* call other procedures to initialize the
/* screen, input the parameters, clean up the
/* old FFT segments from the mailboxes, set up
/* new segments, create the tasks, and then wait
/* for messages from the operator (abort) or
/* other tasks (FFT or OUTPUT done).
/*****
550 1 SUPERVISOR_TASK: PROCEDURE PUBLIC;
551 2 call initialize_buffers;
552 2 call rq$end$init$task;
553 2 call initial screen;
554 2 call initialize_tasks;
555 2 do forever;
556 3 call input_parameters;
557 3 call initialize_segments;
558 3 call monitor_mailboxes;
559 3 end;
560 2 END SUPERVISOR_TASK;
561 1 END SUPERVISOR_MODULE;

MODULE INFORMATION:
CODE AREA SIZE = 1032H 4146D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 0084H 132D
MAXIMUM STACK SIZE = 0024H 36D
1197 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION

```



```

ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE INPUT_TASK_MODULE
OBJECT MODULE PLACED IN :F1:input.OBJ
COMPILER INVOKED BY: plm86 :F1:input.p86
/*
 * $title('INPUT TASK FOR AP NOTE 110, OCTOBER 1980')
 * $large debug
 * INPUT_TASK_MODULE:
 * do;
 * $include(:f1:nucprm.ext)
 * = $SAVE NOLIST
 */
89 1 declare token; literally 'word';

/* The following two tokens, the FFT sample
/* segment format, and the root job directory
/* form the entire interface for this task with
/* the rest of the system. */

90 1 declare to_input_mbx token external;
91 1 declare to_fft_mbx token external;

92 1 declare ascii_mask; literally '30H';
93 1 declare carriage_return; literally '0DH';
94 1 declare done; literally 'OFFH';
95 1 declare first_loop; literally 'OFFH';
96 1 declare forever; literally 'while 1';
97 1 declare frames_to_process_entry; literally '50';
98 1 declare hardware_interrupt_level_3; literally '0038H';
99 1 declare interrupt_task_created; literally 'OFFH';
100 1 declare interrupt_task_not_created; literally '00H';
101 1 declare latch_the_data; literally '040H';
102 1 declare line_feed; literally '0AH';
103 1 declare new_fft_run; literally 'OFFH';
104 1 declare no_response_requested; literally '00H';
105 1 declare no_data_segment; literally '00H';
106 1 declare not_done; literally '00H';
107 1 declare not_first_loop; literally '00H';
108 1 declare not_valid; literally '00H';
109 1 declare null; literally '00H';
110 1 declare processed_so_far_entry; literally '33';
111 1 declare queue_fifo; literally '00H';
112 1 declare rootjob; literally '03H';
113 1 declare run_continuous; literally 'OFFFH';
114 1 declare sample_LSB; literally '0081H';
115 1 declare sample_MSB; literally '0080H';
116 1 declare size_2_bytes; literally '2';
117 1 declare size_120_bytes; literally '120';
118 1 declare supervisor_job; literally '00H';
119 1 declare th_write; literally '05';
120 1 declare this_is_the_interrupt_task; literally '01H';
121 1 declare timer_one_port; literally '00D2H';
122 1 declare timer_mode_control_port; literally '00D6H';
123 1 declare valid; literally 'OFFH';

```

```

124 1 declare wait_forever          literally 'OFFFHH';

125 1 declare data_segment_token      token;
126 1 declare dummy_mbx               token;
127 1 declare from_interrupt_task_mbx token;
128 1 declare handler_dummy_mbx       token;
129 1 declare handler_status          token;
130 1 declare interrupt_status        token;
131 1 declare interrupt_task_token     token;
132 1 declare interrupt_message_token token;
133 1 declare output_buffer_token     token;
134 1 declare return_mbx              token;
135 1 declare root_job_token           token;
136 1 declare signal_interrupt_token  token;
137 1 declare status                  token;
138 1 declare to_interrupt_task_mbx   token;
139 1 declare th_out_mbx              token;

140 1 declare sample_data            integer;

141 1 declare sample_input_data structure(
      LSB      byte,
      MSB      byte) at (@sample_data);

142 1 declare current_timer_value     word;

143 1 declare timer_values structure(
      LSB      byte,
      MSB      byte) at (@current_timer_value);

144 1 declare done_flag               byte;
145 1 declare first_input_loop_flag   byte;
146 1 declare frames_received         byte;
147 1 declare general_index           byte;
148 1 declare interrupt_task_flag     byte;
149 1 declare sample_valid            byte;

150 1 declare timer_threshold         word;

151 1 declare value_to_convert         word;

152 1 declare converted_value structure(
      first_digit      byte,
      second_digit     byte);

/* The following declare is for the home */
/* characters for the Hazeltine terminals. */
/* The sequence is tilde, DC2. */

153 1 declare cursor_home_chars(2) byte data (07EH,012H);

154 1 declare output_buffer_pointer   pointer;

```

```

155 1      declare output_buffer_pointer_values structure(
              offset      word,
              base        word) at
              (@output_buffer_pointer);

156 1      declare output_buffer based output_buffer_pointer
              structure(
              function     word,
              count        word,
              exception_code word,
              actual       word,
              home_chars(2) byte,
              line_index(24) byte,
              character(80) byte);

157 1      declare data_segment_pointer pointer public;

158 1      declare data_segment_pointer_values structure(
              offset      word,
              base        word) at
              (@data_segment_pointer);

              /* The following is the FFT data segment format. */

159 1      declare data_segment based data_segment_pointer
              structure(
              samples_per_frame      word,
              sample_interval        word,
              frames_to_average      word,
              continuous_flag        word,
              this_frame_number      word,
              number_samples_missed word,
              sample_pointer         word,
              reset_flag             word,
              sample(256)           integer);

160 1      declare input_status_line(80) byte data
              ('      The INPUT TASK has processed ',
              '      frames out of      frames to'
              '      average. ');

161 1      FAST_INPUT_HANDLER:
162 2      PROCEDURE (FFT_SEGMENT_POINTER) EXTERNAL;
163 2      DECLARE FFT_SEGMENT_POINTER POINTER;
              END FAST_INPUT_HANDLER;

              /******
              /* SLOW INPUT HANDLER is an interrupt procedure */
              /* that receives an interrupt when the 8253 */
              /* interval timer counts to zero. The 8253 is */
              /* free running, so it starts counting from the */
              /* top again. The 8253 counter is tested */
              /* in a polling fashion to be sure it reads the */
              /* sample, which resets the conversion, */
              /* at a precise time. This aids in removing */

```

```

/* jitter from the sample intervals. When all */
/* of the samples have been taken, */
/* SLOW INPUT HANDLER calls signal interrupt, */
/* which lets the INTERRUPT_TASK procedure know */
/* the buffer is full. */
/*****

164 1 SLOW_INPUT_HANDLER: PROCEDURE INTERRUPT 59 PUBLIC;

/* First set the sample to not valid (we have */
/* to be past the timer threshold before the */
/* sample becomes valid). */

165 2 sample_valid = not_valid;
166 2 timer_loop:

/* Make the timer value stable and read it. */
output (timer_mode_control_port) = latch_the_data;
167 2 timer_values.LSB = input (timer_one_port);
168 2 timer_values.MSB = input (timer_one_port);

/* If it is not past the threshold, then some */
/* future sample will be valid. */

169 2 if current_timer_value > timer_threshold then
170 2 do;
171 3 sample_valid = valid;
172 3 goto timer_loop;
173 3 end;

/* We get to the else only if we are past the */
/* timer threshold. */
else
174 2 do;
175 3 sample_input_data.LSB = input(sample_LSB);
176 3 sample_input_data.MSB = input(sample_MSB);

/* If the sample is valid, we must have come */
/* in before the threshold so we know we */
/* sampled as close to the right time as */
/* possible. */

177 3 if sample_valid = valid then
178 3 do;

/* However, we want to ignore the first */
/* sample (which was started a long */
/* time ago). */

179 4 if first_input_loop_flag = first_loop then
180 4 first_input_loop_flag = not_first_loop;
else
181 4 do;

```



```

182 5      data segment.sample
          (data segment.sample_pointer)
          = sample_data;
183 5      data segment.sample_pointer
          = data segment.sample_pointer+1;
184 5      end;
185 4      end;
          else
186 3      do;
187 4      data segment.number_samples_missed
          = data segment.number_samples_missed+1;
188 4      data segment.sample_pointer = 0;
189 4      end;
190 3      sample_valid = not_valid;
191 3      end;

/* If we are done, we have to let the */
/* INPUT_TASK know the buffer is full. */
/* Otherwise, we wait for the next interrupt. */

192 2      if data segment.sample_pointer
          >= data segment.samples_per_frame then
193 2      call rq$signal$interrupt
          (hardware_interrupt_level_3,
           @handler_status);
          else
194 2      call rq$exit$interrupt
          (hardware_interrupt_level_3,
           @handler_status);

195 2      END SLOW_INPUT_HANDLER;

/*****
/* INTERRUPT_TASK exists because if an interrupt */
/* task goes to sleep, the level of the */
/* interrupt task, interrupt handler, and lower */
/* levels remain disabled. In order to prevent */
/* this from happening in this application, this */
/* task notifies the INPUT_TASK that the buffer */
/* is full. INPUT_TASK disables Level 3 and */
/* returns the token to INTERRUPT_TASK. */
/* INTERRUPT_TASK will then call wait interrupt, */
/* enabling lower levels. Since INPUT_TASK */
/* disabled Level 3, no Level 3 interrupts will */
/* be serviced until Level 3 is enabled by */
/* INPUT_TASK. */
/* */
/* NOTE THAT PLM/86 REQUIRES THE USE OF THE */
/* BUILT IN INTERRUPT$PTR PROCEDURE TO OBTAIN */
/* THE PROPER INTERRUPT PROCEDURE ENTRY POINT. */
*****/

196 1      INTERRUPT_TASK: PROCEDURE PUBLIC;

```

```

197 2 call rq$set$interrupt
      (hardware_interrupt_level_3,
      this is the interrupt task,
      INTERRUPT$PTR(SLOW INPUT HANDLER),
      no_data_segment, @interrupt_status);
198 2 do forever;
199 3 call rq$wait$interrupt
      (hardware_interrupt_level_3,
      @interrupt_status);
200 3 call rq$send$message
      (from_interrupt_task_mbx,
      interrupt_message_token,
      to_interrupt_task_mbx, @interrupt_status);
201 3 interrupt_message_token = rq$receive$message
      (to_interrupt_task_mbx, wait_forever,
      @dummy_mbx, @interrupt_status);
202 3 end;
203 2 END INTERRUPT_TASK;

      /*****
      /* CONVERT DIGITS is a small procedure for
      /* converting a hex number into an ASCII
      /* number, with the advance knowledge that the
      /* hex number will be less than 99 decimal (in
      /* this case, less than 32 decimal).
      /* *****/

204 1 CONVERT_DIGITS: PROCEDURE;

205 2 converted_value.first_digit = ascii_mask;
206 2 converted_value.second_digit = ascii_mask;
207 2 done_flag = not_done;
208 2 do while done_flag = not_done;
209 3 value_to_convert = value_to_convert - 10;
      /* The problem here is we need to check for
      /* a negative value when we have BYTE values
      /* which are, by definition, positive and
      /* modulo 256. So we adapt by checking for
      /* > 200 decimal, which should mean the
      /* value has "wrapped" around zero. If it
      /* has, we can get our previous value back
      /* by adding 10.
      /*
210 3 if value_to_convert < 200 then
211 3 converted_value.first_digit

```

```

                                = converted_value.first_digit + 1;
                                else
212   3   do;
213   4   value_to_convert = value_to_convert + 10;
214   4   converted_value.second_digit
                                = converted_value.second_digit +
                                value_to_convert;
215   4   done_flag = done;
216   4   end;
217   3   end;
218   2   END CONVERT_DIGITS;

/* ***** */
/* SEND STATUS converts the current frame */
/* number into ASCII and stuffs it into the */
/* previously initialized status line. Then */
/* SEND STATUS sends the status line to the */
/* terminal handler and waits for the segment */
/* to be returned. */
/* ***** */

219   1   SEND_STATUS: PROCEDURE;
220   2   value_to_convert = data_segment.this_frame_number;
221   2   call convert_digits;
222   2   output_buffer.character (processed_so_far_entry)
                                = converted_value.first_digit;
223   2   output_buffer.character (processed_so_far_entry+1)
                                = converted_value.second_digit;

224   2   value_to_convert = data_segment.frames_to_average;
225   2   call convert_digits;
226   2   output_buffer.character (frames_to_process_entry)
                                = converted_value.first_digit;
227   2   output_buffer.character (frames_to_process_entry+1)
                                = converted_value.second_digit;

228   2   call rq$send$message
                                (th_out_mbx, output_buffer_token,
                                return_mbx, @status);
229   2   output_buffer_token = rq$receive$message
                                (return_mbx, wait_forever,
                                @dummy_mbx, @status);

230   2   END SEND_STATUS;

/* ***** */
/* INPUT_DATA selects the fast or slow input */
/* handler, initializes the 8253 timer as */
/* necessary, and calls the appropriate input */

```

```

/* handler. Please note the values of the */
/* intervals selected for sampling are */
/* scaled by 60/64 so the actual frequency */
/* output of the 128 sample FFT algorithm will */
/* match up with the base 10 x axis labels. */
/* Base 10 doesn't map too well to a binary */
/* x-axis that runs from 1 to 64. */
/*****

231 1 INPUT_DATA: PROCEDURE;

232 2 declare LSB_120Hz_interval literally '058H';
233 2 declare MSB_120Hz_interval literally '002H';
234 2 declare LSB_600Hz_interval literally '078H';
235 2 declare MSB_600Hz_interval literally '000H';

/* The 8253 timer is running at 143.6 KHz, or */
/* 6.5 microseconds per count. We have to */
/* restart the sampling process precisely, so */
/* we count down after the interrupt to be */
/* sure we are synchronized. In this case, */
/* we have a 300 microsecond window after the */
/* interrupt to get the sample. 300 */
/* microseconds is roughly 42 times 6.5 */
/* microseconds. */

236 2 declare threshold_for_120Hz literally '022EH';
237 2 declare threshold_for_600Hz literally '0048H';

238 2 declare a_3906_microsecond_interval
239 2 declare five_places literally '0F42H';
240 2 declare nucleus_allocated_stack literally '5';
241 2 declare shift_integer_right literally '00H';
242 2 declare software_priority_level_0 literally '00H';
243 2 declare software_priority_level_66 literally '66';
244 2 declare stack_size_512 literally '512';
245 2 declare task_flags literally '00H';
246 2 declare this_task literally '00H';
247 2 declare timer_mode_control_word literally '74H';

248 2 declare enable_conversion literally '00';

249 2 declare input_command literally '0080H';

/* The first thing we do is start the conver- */
/* sions. We don't care about the first */
/* data since we are going to ignore it. Each */
/* time we read both bytes of the present */
/* converted value, we start the next */
/* conversion. This initialization will */
/* prepare for the real data gathering. */

250 2 output(input_command) = enable_conversion;

```



```

251 * 2   if data_segment.sample_interval > 391 then
252 * 2       do;
253 * 3       output (timer_mode_control_port)
                = timer_mode_control_word;
254 * 3       if data_segment.sample_interval
                = a_3906_microsecond_interval then
255 * 3           do;
256 * 4               timer_threshold
                    = threshold_for_120Hz;
257 * 4               output (timer_one_port)
                    = LSB_120Hz_interval;
258 * 4               output (timer_one_port)
                    = MSB_120Hz_interval;
259 * 4           end;
                else
260 * 3           do;
261 * 4               timer_threshold
                    = threshold_for_600Hz;
262 * 4               output (timer_one_port)
                    = LSB_600Hz_interval;
263 * 4               output (timer_one_port)
                    = MSB_600Hz_interval;
264 * 4           end;
265 * 3       first_input_loop_flag = first_loop;

266 * 3       if interrupt_task_flag
                = interrupt_task_not_created then
267 * 3           do;
268 * 4               interrupt_task_token = rq$create$task
                    (software_priority_level_66,
                     @interrupt_task, no_data_segment,
                     nucleus_allocated_stack,
                     stack_size_512, task_flags, @status);
269 * 4               call rq$catalog$object
                    (supervisor_job, interrupt_task_token,
                     @(12,'INTERRUPTTSK'), @status);
270 * 4               interrupt_task_flag
                    = interrupt_task_created;

271 * 4           end;
272 * 3       else call rq$enable
                    (hardware_interrupt_level_3, @status);

                /* Now we wait until the slow handler */
                /* fills the buffer. */
273 * 3       signal_interrupt_token = rq$receive$message
                    (from_interrupt_task_mbx, wait_forever,
                     @dummy_mbx, @status);

                /* If we get the token, we know the buffer */
                /* is full, so we disable level 3 */

```

```

274 3      call rq$disable
           (hardware_interrupt_level_3, @status);

           /* And return the token so the
           /* INTERRUPT_TASK can enable lower
           /* interrupt levels. */

275 3      call rq$sendMessage
           (to_interrupt_task_mbx,
            signal_interrupt_token,
            no_response_requested, @status);

276 3      end;
           else
277 2      do;
           /* The fast INPUT handler must sample at
           /* precise intervals that do not allow
           /* variable interrupt latency. Therefore
           /* we raise the priority level to 0--the
           /* highest--and just sample in a polling
           /* fashion until the buffer is filled. */

278 3      call rq$set$priority
           (this_task, software_priority_level_0,
            @status);

279 3      call FAST_INPUT_HANDLER (data_segment_pointer);
280 3      call rq$set$priority
           (this_task, software_priority_level_66,
            @status);

281 3      end;

282 2      do general_index = 0 to 127;
283 3      data_segment.sample (general_index)
           = shift integer_right
           (data_segment.sample (general_index),
            five_places);

284 3      end;

285 2      do general_index = 128 to 255;
286 3      data_segment.sample (general_index) = 0000H;
287 3      end;

288 2      END INPUT_DATA;

           /******
           /* UPDATE_FRAME_NUMBER just updates the frame
           /* number parameter on the data segments.
           /******

289 1      UPDATE_FRAME_NUMBER: PROCEDURE;

290 2      data_segment.number_samples_missed = 0;
291 2      data_segment.sample_pointer = 0;

```

```

292 2      if frames_received = data_segment.frames_to_average
          then frames_received = 0;

294 2      if data_segment.reset_flag = new_fft_run then
295 2      do;
296 3          frames_received = 0;
297 3          data_segment.reset_flag = 0;
298 3      end;

299 2      frames_received = frames_received + 1;
300 2      data_segment.this_frame_number = frames_received;

301 2      END UPDATE_FRAME_NUMBER;

          /*****
          /* INITIALIZE_BUFFERS takes care of the usual */
          /* trivia of setting up the pointers, creating */
          /* the return mailbox, looking up the terminal */
          /* handler, and all that other small garbage. */
          *****/

302 1      INITIALIZE_BUFFERS: PROCEDURE;

303 2          return_mbx = rq$create$mailbox
          (queue_fifo, @status);
304 2          call rq$catalog$object
          (supervisor_job, return_mbx,
            @(9, 'I RET MBX'), @status);

305 2          from_interrupt_task_mbx = rq$create$mailbox
          (queue_fifo, @status);
306 2          call rq$catalog$object
          (supervisor_job, from_interrupt_task_mbx,
            @(12, 'FM INTSK MBX'), @status);

307 2          to_interrupt_task_mbx = rq$create$mailbox
          (queue_fifo, @status);
308 2          call rq$catalog$object
          (supervisor_job, to_interrupt_task_mbx,
            @(12, 'TO INTSK MBX'), @status);

309 2          interrupt_message_token = rq$create$segment
          (size 2_bytes, @status);
310 2          call rq$catalog$object
          (supervisor_job, interrupt_message_token,
            @(10, 'INTT SK MSG'), @status);

311 2          interrupt_task_flag
          = interrupt_task_not_created;

312 2          output_buffer_token = rq$create$segment
          (size 120 bytes, @status);
313 2          call rq$catalog$object
          (supervisor_job, output_buffer_token,
            @(10, 'I BUFF SEG'), @status);

```

```

314 2      output_buffer_pointer_values.offset    = 0;
315 2      output_buffer_pointer_values.base
          = output_buffer_token;
316 2      output_buffer.function = th write;
317 2      output_buffer.count    = 110;
318 2      do general_index = 0 to 6;
319 3          output_buffer.home_chars (general_index)
          = cursor_home_chars (general_index);
320 3      end;
321 2      do general_index = 0 to 21;
322 3          output_buffer.line_index (general_index)
          = line_feed;
323 3      end;
324 2      do general_index = 22 to 23;
325 3          output_buffer.line_index (general_index)
          = null;
326 3      end;
327 2      do general_index = 0 to 78;
328 3          output_buffer.character (general_index)
          = input_status_line (general_index);
329 3      end;
330 2      root_job_token = rq$get$task$tokens .
          (rootjob, @status);
331 2      th_out_mbx
          = rq$lookup$object
          (root_job_token, @(11,'RQTHNORMOUT'),
          wait_forever, @status);
332 2      frames_received = 0;
333 2      END INITIALIZE_BUFFERS;

          /*****
          /* The actual INPUT_TASK begins here. It      */
          /* initializes the buffers to begin things,    */
          /* then waits forever for the FFT sample      */
          /* segment. It then samples the data, fills   */
          /* the FFT data segment, and sends it to the  */
          /* FFT TASK. The INPUT_TASK then updates its  */
          /* status line, sends it to the terminal     */
          /* handler, and returns to the mailbox to    */
          /* wait forever.                             */
          *****/

334 1      INPUT_TASK: PROCEDURE PUBLIC;
335 2      call initialize_buffers;
336 2      data_segment_pointer_values.offset = 0;
337 2      do forever;

```



```

/* Wait forever for an FFT data segment at */
/* the to_input_mbx. */
338 3 data_segment_token = rq$receive$message
      (to_input_mbx, wait_forever,
      @dummy_mbx, @status);
339 3 data_segment_pointer_values.base
      = data_segment_token;

340 3 call update_frame_number;
341 3 call input_data;
342 3 call send_status;

343 3 call rq$send$message
      (to_fft_mbx, data_segment_token,
      no_response_requested, @status);

344 3 end;

345 2 END INPUT_TASK;
346 1 END INPUT_TASK_MODULE;

```

MODULE INFORMATION:

```

CODE AREA SIZE = 070BH 1803D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 0036H 54D
MAXIMUM STACK SIZE = 002AH 42D
754 LINES READ
0 PROGRAM ERROR(S)
END OF PL/M-86 COMPILATION

```

```

*****
/* The actual INPUT TASK begins here. It
/* initializes the buffers to begin things,
/* then waits forever for the FFT sample
/* segment. It then samples the data, fills
/* the FFT data segment, and sends it to the
/* FFT TASK. The INPUT TASK then updates its
/* status line, sends it to the terminal
/* handler, and returns to the mailbox to
/* wait forever.
*****

```

```

INPUT_TASK: PROCEDURE PUBLIC;
1
call initialize_buffers;
2
data_segment_pointer_values.offset = 0;
3
do forever;
4

```

MCS-86 MACRO ASSEMBLER FSTINP
 ISIS-II MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE FSTINP
 OBJECT MODULE PLACED IN :F1:FSTINP.OBJ
 ASSEMBLER INVOKED BY: asm86 :f1:fstinp.a86

LOC OBJ LINE SOURCE

```

1      ;*****
2      ;
3      ; FAST_INPUT_HANDLER for APNOTE 110,
      ; OCTOBER 1980
4      ;
5      ; FAST_INPUT_HANDLER is an assembler routine
      ; that runs at priority
6      ; level 0 and simply drives an analog to
      ; digital convertor and
7      ; stuffs the samples into a data segment until
      ; all of the samples
8      ; have been taken. FAST_INPUT_HANDLER has
      ; passed to it the address
9      ; of the data segment, in which the offset is
      ; known to be zero.
10     ; FAST_INPUT_HANDLER returns nothing to the
      ; calling routine.
11     ;
12     ; FAST_INPUT_HANDLER provides the proper timing
      ; for sampling at a
13     ; 39, 78, and 391 microsecond intervals using
      ; timed loops of
14     ; software instructions. In order to provide
      ; an FFT without large
15     ; amounts of jitter, the sample intervals must
      ; be uniform in time.
16     ; iRMX 86 cannot guarantee this uniformity due
      ; to its real time
17     ; design, so this routine takes complete
      ; control of the processor
18     ; for the (39 times 128) 4.9 milliseconds or
      ; (78 times 128) 9.9
19     ; or (391 times 128) 50 milliseconds required
      ; to complete a frame
20     ; of 128 samples.
21     ;
22     ; iAPX 86 register useage is the following:
23     ;
24     ; AX -- general          BX -- stack index
25     ; CX -- loop delay counter  DX -- sample value
26     ;
27     ; BP -- stack
28     ; DI -- offset index into FFT SI -- not used
29     ; data segment
30     ;
31     ; DS -- base for FFT data ES -- not used
32     ; segment

```

```

33      ;
34      ;*****
35      ;
36      ;
37      ASSUME DS:FAST_INPUT_DATA, SS:STACK,
          CS:FAST_INPUT_CODE, ES:NOTHING
38      ;
39      PUBLIC FAST_INPUT_HANDLER
40      ;
0080 41      SAMPLE_LSB EQU 0080H
0081 42      SAMPLE_MSB EQU 0081H
000E 43      FIRST_PASS EQU 14
0002 44      SAMPLE_INCREMENT EQU 2
0002 45      SAMPLE_INTERVAL EQU 2
010E 46      SAMPLE_MAX EQU 270
47      ;
48      ;
---- 49      FAST_INPUT_DATA SEGMENT WORD
          PUBLIC 'DATA'
50      ;
0000 51      LOOP_VALUE DW ?
52      ;
---- 53      FAST_INPUT_DATA ENDS
54      ;
55      ;
---- 56      STACK SEGMENT STACK 'STACK'
57      ;
0000 58      (20 DW 20 DUP(0)
          0000
          )
59      ;
---- 60      STACK ENDS
61      ;
62      ;
---- 63      FAST_INPUT_CODE SEGMENT PARA
          PUBLIC 'CODE'
64      ;
0000 65      FAST_INPUT_HANDLER PROC FAR
66      ;
0000 67      1E PUSH DS
0001 68      55 PUSH BP
          ; SAVE BP IN STACK
0002 69      8BEC MOV BP, SP
          ; SET BP TO STACK POINTER
0004 70      8E5E0A MOV DS, [BP + 10]
          ; PUT BASE OF SAMPLE SEGMENT IN DS
0007 71      BF0200 MOV DI, SAMPLE_INTERVAL
          ; DX IS USED TO INDEX INTO THE DATA SEGMENT
000A 72      8B05 MOV AX, DS:[DI]
          ; SET AX TO SAMPLE_INTERVAL PARAMETER
000C 73      BF0E00 MOV DI, FIRST_PASS
          ; RESET DI TO FIRST SAMPLE - 14
000F 74      3D2700 CMP AX, 39
          ; IF AX = 39, SET LOOP_VALUE TO 9--LOOP
0012 75      740E JZ SET_39_US

```

```

; TAKES ABOUT 3 US PER DECREMENT
0014 3D4E00 76 CMP AX, 78
; IF AX = 78, SET LOOP_VALUE TO 22--BASIC
0017 7412 77 JZ SET 78 US
; CYCLE IS 13 US, PLUS (22 X 3) = 79 US
0019 C70600007E00 R 78 MOV LOOP_VALUE, 126
; 391 IS ONLY ONE LEFT-13 + (126X3)
= 391
001F EB1090 79 JMP INPUT_LOOP ;
0022 C70600000900 R 80 SET 39 US: MOV LOOP_VALUE, 9
; TIMING IS BY SOFTWARE--SET
DELAY COUNT
0028 EB0790 81 JMP INPUT_LOOP ;
002B C70600001600 R 82 SET 78 US: MOV LOOP_VALUE, 22 ;
0031 8B0E0000 R 83 INPUT_LOOP: MOV CX, LOOP_VALUE
; USE CX TO KEEP TRACK OF DELAY
0035 E480 84 IN AL, SAMPLE_LSB
; SET AL TO LSB OF INPUT SAMPLE
0037 8AD0 85 MOV DL, AL
; PUT THE LSB IN DL (8 BIT XFERS ONLY)
0039 E481 86 IN AL, SAMPLE_MSB
; SET AL TO MSB OF INPUT SAMPLE
; THIS RESTARTS SAMPLE PROCESS
003B 8AF0 87 MOV DH, AL
; PUT AL IN DH TO COMPLETE THE VALUE
003D 83FF0E 89 CMP DI, FIRST_PASS
; WE WANT TO SKIP THE FIRST SAMPLE
0040 7408 90 JZ SKIP_INPUT ;
0042 83C702 91 ADD DI, SAMPLE_INCREMENT
; INCREMENT DI BY 2
0045 8915 92 MOV DS:[DI], DX
; PUT SAMPLE DATA IN SEGMENT
0047 EB0990 93 JMP DELAY
; AND JUMP TO SOFTWARE DELAY LOOP
004A 83C702 94 SKIP_INPUT: ADD DI, SAMPLE_INCREMENT
; INCREMENT DI BY 2
004D 90 95 NOP
; AND NOP FIVE TIMES FOR EVEN TIMING
004E 90 96 NOP ;
004F 90 97 NOP ;
0050 90 98 NOP ;
0051 90 99 NOP ;
0052 90 100 DELAY: NOP
; THIS NOP ADDS 3 CLOCKS PER DECREMENT
0053 E0FD 101 LOOPNZ DELAY
; DEC CX AND LOOP--1.5 US PER DECREMENT
0055 81FF0E01 102 CMP DI, SAMPLE_MAX
; COMPARE DI TO SEE IF WE ARE DONE
0059 75D6 103 JNE INPUT_LOOP
; IF NOT, GO BACK FOR ANOTHER SAMPLE
005B 5D 104 POP BP
; OTHERWISE POP BP, DS, AND RETURN
005C 1F 105 POP DS ;
005D CA0400 106 RET 4H ;
107 ;

```



```

108 FAST_INPUT_HANDLER ENDP
109 ;
110 FAST_INPUT_CODE ENDS
111 ;
112 END
; CYCLE 18 US; 18 X 2 = 36 US
; MOV LOOP VALUE, 18
0019 C700007E00 R 78 MOV LOOP VALUE, 18
001F EB1090 JMP INPUT_LOOP
0023 C700000000 R 80 SET 39 US; MOV LOOP VALUE, 9
; TIMING IS BY SOFTWARE--SET
; DELAY COUNT
0023 EB0790 JMP INPUT_LOOP
002B C7000001500 R 82 SET 78 US; MOV LOOP VALUE, 37
0031 8B0E0000 R 83 INPUT_LOOP; MOV CX, LOOP VALUE
; USE CX TO KEEP TRACK OF DELAY
0033 E480 IN AL, SAMPLE_LSB
; SET AL TO LSB OF INPUT SAMPLE
0037 84D0 MOV DL, AL
; BUT THE LSB IN DL (8 BIT XPIRS ONLY)
0039 E481 IN AL, SAMPLE_MSB
; SET AL TO MSB OF INPUT SAMPLE
; THIS RESTARTS SAMPLE PROCESS
003B 84F0 MOV BH, AL
; BUT AL IN BH TO COMPLETE THE VALUE
003D 84F0E0 CMP DL, FIRST_PASS
; WE WANT TO SKIP THE FIRST SAMPLE
0040 7408 JB SKIP_INPUT
0042 83C702 91 ADD DI, SAMPLE_INCREMENT
; INCREMENT DI BY 2
0044 8412 MOV DS:DI, CX
; BUT SAMPLE DATA IN SEGMENT
0047 EB0990 JMP DELAY
; AND JUMP TO SOFTWARE DELAY LOOP
004A 83C702 94 SKIP_INPUT; ADD DI, SAMPLE_INCREMENT
; INCREMENT DI BY 2
004D 90 NOP
; AND NOP FIVE TIMES FOR EVEN TIMING
004E 90 NOP
004F 90 NOP
0050 90 NOP
0051 90 NOP
0052 90 DELAY; NOP
; THIS NOP ADDS 3 CLOCKS PER DECREMENT
0053 E0F0 101 LOOPNZ DELAY
; DEC CX AND LOOP--1.5 US PER DECREMENT
0055 B1F0E01 103 CMP DI, SAMPLE_MAX
; COMPARE DI TO SEE IF WE ARE DONE
0059 75D4 103 JNE INPUT_LOOP
; IF NOT, GO BACK FOR ANOTHER SAMPLE
005B 5D 104 POP BP
; OTHERWISE POP BP, DS, AND RETURN
005C 1F 105 POP DS
005D C40400 106 RET 4H
107

```

Both the RAM and ROM-based configurations will be discussed in this appendix. They are essentially identical processes. In either case, the first step is to define a map of system memory. Once the map is known, the following sequence is suggested for locating code in memory:

- 1) Reserve memory 0H to 03FFH for the Nucleus interrupt vector.
- 2) If the system is RAM based and the code is loaded by the iSBC 957A Monitor, reserve locations 03FFH to 07FFH for the monitor's use.
- 3) Configure each of the necessary portions of the iRMX 86 Operating System and locate them sequentially in memory.
- 4) For a RAM-based development system, allow 2K of RAM for the system Root Job. Placing the Root Job after the portions of the iRMX 86 Operating System, which are relatively fixed in size during development, and before the development code will give the Root Job a fixed address. This will prevent having to move the Root Job and reconfigure the system when the development code grows. For final EPROM-based systems, the Root Job should be placed after the development code.
- 5) Link and locate each of the application code modules sequentially in memory.
- 6) Define the RAM available to the system.
- 7) Define memory NOT available to the system. This includes application code, EPROM, and non-existent memory within the 1 megabyte address space.
- 8) Create the configuration file using the address maps produced by the locate steps and the memory map defined in steps 6 and 7.
- 9) Create the Root Job from this configuration file.
- 10) Load and test the system in RAM.
- 11) If the system has been fully debugged, load the code into EPROM and test the final system.

The above steps are necessary for both the RAM development system and the final EPROM system. Converting this application from RAM to EPROM requires reconfiguring the Nucleus to include only those systems calls required by the application, substituting the Terminal Handler Job for the Debugger Job, removing any remaining system calls to catalog objects for debugging, and remapping the system to the EPROM address space. The memory maps for the development and final application are shown in Figures C-1 and C-2.

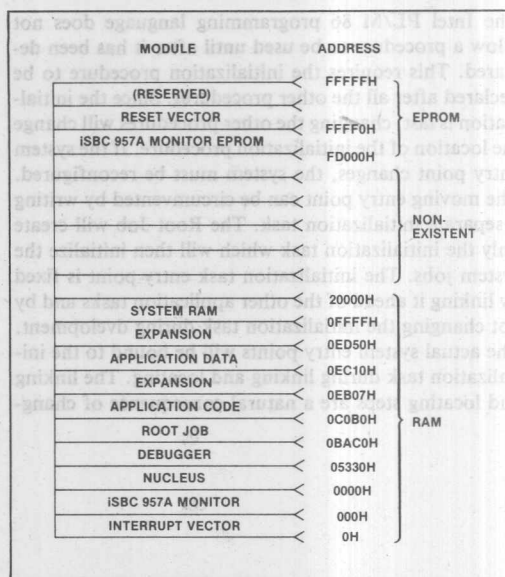


Figure C-1. Development System Memory Map

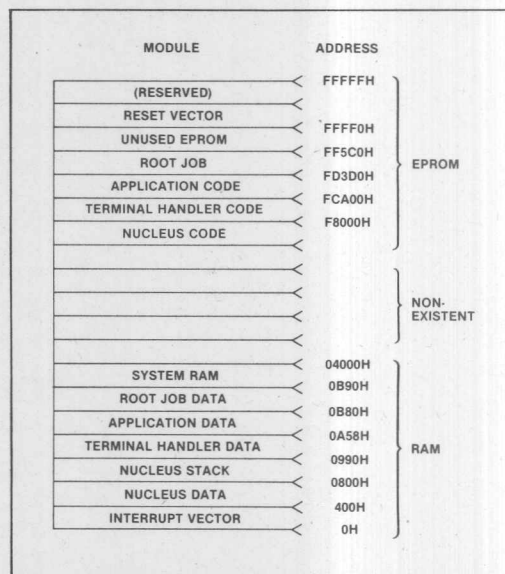


Figure C-2. Final System Memory Map

System configuration is a straightforward but exacting process. As with any such processes, there are some hints that can make development easier. In addition to care in locating the Root Job in memory, users should fix the initialization job entry point and the data RAM addresses.

The Intel PL/M 86 programming language does not allow a procedure to be used until after it has been declared. This requires the initialization procedure to be declared after all the other procedures. Since the initialization is last, changing the other procedures will change the location of the initialization procedure. If the system entry point changes, the system must be reconfigured. The moving entry point can be circumvented by writing a separate initialization task. The Root Job will create only the initialization task which will then initialize the system jobs. The initialization task entry point is fixed by linking it ahead of the other application tasks and by not changing the initialization task during development. The actual system entry points will be bound to the initialization task during linking and locating. The linking and locating steps are a natural consequence of changing

the application code, so binding the fixed system entry point is done automatically during development. The fixed initialization task entry point is used in the configuration file, giving the Root Job an unchanging system entry point.

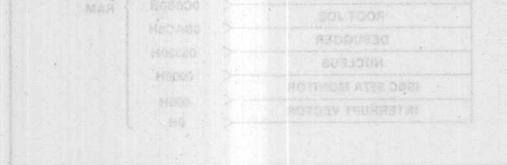


Figure C-1. Development System Memory Map

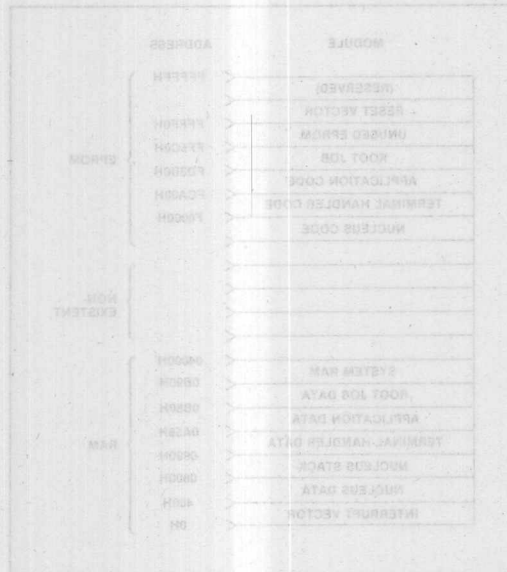


Figure C-2. Final System Memory Map

System configuration is a straightforward but exacting process. As with any such process, there are some things that can make development easier. In addition to care in locating the Root Job in memory, users should fix the initialization job entry point and the data RAM addresses.

The remaining moving target during development is the RAM area for data and stack use. If the data and stack RAM is located before or after the application code, with enough extra memory in between for growth during development, the data and stack locations can stay constant. Fixing both the application entry point and the locations of the stack and data segments will allow development of the application code to proceed without requiring frequent reconfigurations.

1) Reserve memory 0H to 0FFF for the Nucleus in-
2) For a RAM-based development system, allow 2K of RAM for the system Root Job. Placing the Root Job after the portions of the I/OX 86 Operating System, which are relatively fixed in size during development, and before the development code will give the Root Job a fixed address. This will prevent having to move the Root Job and reconfigure the system when the development code grows. For final EPROM-based systems, the Root Job should be placed after the development code.

3) Link and locate each of the application code modules sequentially in memory.

4) Define the RAM available to the system.

5) Define memory NOT available to the system. This includes application code, EPROM, and non-existent memory within the 1 megabyte address space.

6) Create the configuration file using the address maps produced by the locate steps and the memory maps defined in steps 4 and 5.

7) Create the Root Job from this configuration file.

8) Load and test the system in RAM.

9) If the system has been fully debugged, load the code into EPROM and test the final system.

The above steps are necessary for both the RAM development system and the final EPROM system. Converting this application from RAM to EPROM requires reconfiguring the Nucleus to include only those system calls required by the application, substituting the Terminal Handler Job for the Debugger Job, removing any remaining system calls to catalog objects for debugging, and reassigning the system to the EPROM address space. The memory maps for the development and final application are shown in Figures C-1 and C-2.

Introducing the RMX/86™ realtime, multitasking, 16-bit operating system



Now we have a situation in which electronic intelligence is spreading to more and more places, and in addition, the application sophistication at each of these places is growing as well. So when we multiply what it takes to program all the new microcomputer products so that they can be applied, and calculate what that means in terms of how many computer programs will be needed by 1990, we come out with a requirement for over one million computers given before the New York Society of Security Analysts, January 24, 1980.

August 1980

ponents, such systems when needed in OEM microcomputer applications, these problems—tasks can be OEM afford means of effort to develop the minimum familiarity with the hardware that's needed to design executive software. But with Intel's RMX/86 system, you won't have to.

In addition, this second-generation 16-bit system adds error handling flexible command-line decoder, and other advanced OS capabilities to previous options available on the mature RMX/80 package.

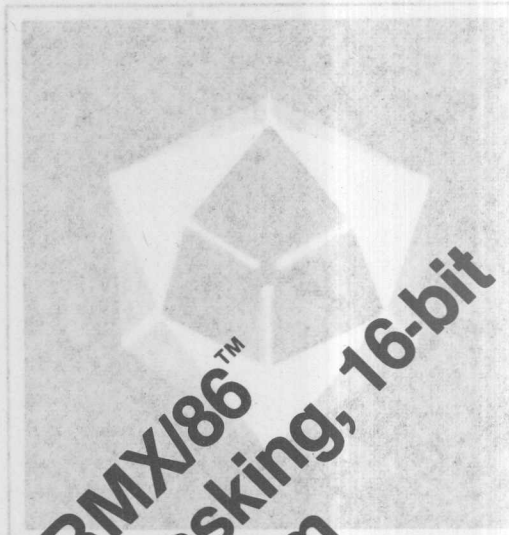
All real-time multitasking systems require executive software to manage the resources shared by the programs (CPU time, memory and I/O), respond to interrupts and then allocate the resources according to established priorities. Normally, these functions are intermingled in an OS with higher-level system functions that often prove awkward in microcomputer applications.

The RMX/86 package combines all these required executive functions in a single module called the nucleus. Other modules in the package tailor the executive system to its application by adding higher-level operating system functions such as disk-file systems. The application programs and optional modules can connect to the nucleus with simple software interfaces.

Since the nucleus is essentially open-ended, it serves the software foundation for the expanding body of applications. The software foundation is the expanding body of applications. The software foundation is the expanding body of applications. The software foundation is the expanding body of applications.

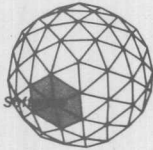
The OEM way

The RMX/86 system fits in with the OEM way of doing business. It's a system software that's easy to use, easy to learn, and easy to integrate with existing systems. It's a system software that's easy to use, easy to learn, and easy to integrate with existing systems. It's a system software that's easy to use, easy to learn, and easy to integrate with existing systems.



Introducing the RMX/86™ Realtime, Multitasking, 16-bit Operating System

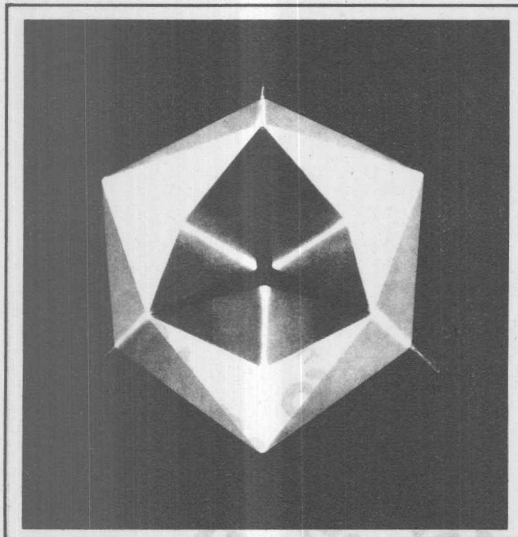
Preview Magazine, May/June 1980



PRODUCT ANNOUNCEMENT

Introducing the RMX/86™ realtime, multitasking, 16-bit operating system

"Now we have a situation in which electronic intelligence is spreading to more and more places, and in addition, the application sophistication at each of these places is growing as well. So, when we multiply what it takes to program all the new microcomputer products so that they can be applied, and calculate what that means in terms of how many computer programmers will be needed by 1990, we come out with a requirement for over one million computer programmers!" Excerpted from a speech by Andrew S. Grove, President, Intel Corporation, given before the New York Society of Security Analysts, January 24, 1980.



As production technologies such as VLSI (Very Large Scale Integration) proliferate—with their greater levels of density—the use of low-cost microcomputer hardware to solve increasingly complex application problems has created a crisis in software. A partial solution to the programmer shortage would be the creation of "building blocks" for system development. It is this modular, building block concept upon which the RMX/86 Operating System is constructed.

The RMX/86™ operating system

This modular operating system for Intel 16-bit microcomputers allows custom operating systems to be assembled largely from off-the-shelf software com-

ponents. Such systems when needed in OEM microcomputer applications, pose problems—rarely can an OEM afford man-years of effort to develop the intimate familiarity with the hardware that's needed to design executive software. But with Intel's RMX/86 system he won't have to.

In addition, this second-generation 16-bit system adds error handling flexible command-line decode, and other advanced OS capabilities to previous options available on the mature RMX/80 package.

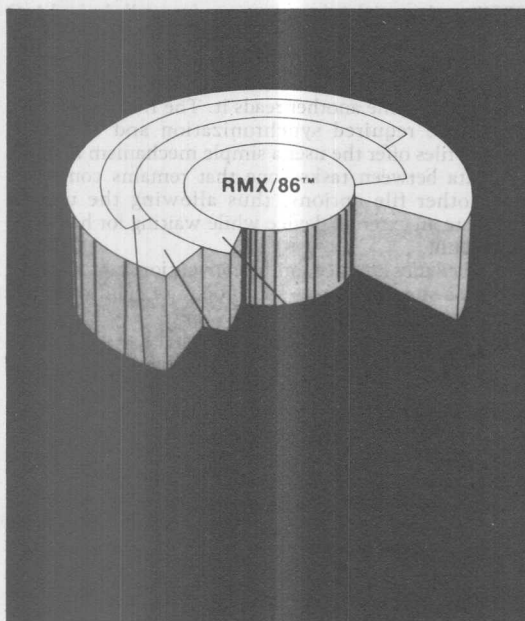
All real-time multitasking systems require executive software to manage the resources shared by the programs (CPU time, memory and I/O), respond to interrupts and then allocate the resources according to established priorities. Normally, these functions are intermingled in an OS with higher-level system functions that often prove superfluous in microcomputer applications.

The RMX/86 package combines all those required executive functions in a single module, called the nucleus. Other modules in the package tailor the executive system to its application by adding higher-level operating system functions such as disk-file systems. The application programs and optional modules connect to the nucleus with simple software interfaces.

Since the nucleus is essentially open-ended, it serves as the software foundation for expanding both higher-level operating system functions and varieties of application programs. Although most microcomputer applications are dedicated, many do require such higher-level capabilities.

The OEM way

The modular approach to system software fits in with the way most OEMs (and high-volume end users) apply single-board computers. They generally start with a minimum amount of hardware marketed at the lowest cost possible. Then, when their users are fully utilizing the original functions and are willing to buy more, the



The RMX/86 Nucleus is the heart of the operating system and is surrounded by application, human interface and input/output facilities.

OEM adds functionality by increasing the executive facilities, application tasks and hardware—allowing the fastest turnaround possible.

To see how the RMX/86 operating system works, consider a typical example. Say an OEM develops a factory heating and air-conditioning control system with a single-board computer, analog I/O, special control devices and operator terminal. He uses the nucleus, and terminal handler modules, plus user tasks stored in EPROM.

But suppose the OEM's customer wants to enhance the system with, say, disk storage. He simply adds a disk controller board and I/O system software. The original programs could also be made disk-resident. If the original application had been based on a conventional executive, extensive redevelopment would have been necessary.

If, on the other hand, the application had used the full I/O system from the start, initial sales would have suffered because the OEM's system would have been more costly. And if the software is not extendible, future sales are lost.

The historic problem with conventional "general-purpose" operating systems is that they usually depend on hardware that the application would not otherwise need—for example, large disk systems while a typical OEM system uses no mass storage at all. Modular operating systems are designed to accommodate a diversity of applications without undue hardware imposed on them.

For an even closer fit with customers applications,

the RMX/86 modules closely parallel hardware modularity.

Corresponding to the hardware, the RMX/86 nucleus manages CPU, memory and I/O sources. When linked to optional modules it can support standard iSBC devices like consoles and disk controllers. Similarly, users may add device driver software modules for their custom peripherals. All software can reside in EPROM/ROM if mass storage is not available.

The RMX/86 operating system is designed for configuration, to user specification, on an Inteltec development system. The same system supports application software development as well as linking and locating both high-level and assembly language.

What does a nucleus do?

A real-time multitasking nucleus gives a program the means to monitor and control external events. Tasks running concurrently, using the services of the executive are signaled with interrupts and the executive schedules resources for the tasks on the basis of priority. For example, a task trying to bring an oven's temperature under control should have a much higher priority than a report generating task. The nucleus' scheduler decides whether a running task should be interrupted to process data from an interrupting device.

The RMX/86 nucleus makes event-driven priority scheduling possible by monitoring system states, determining task requirements, allocating the resources and gathering the resources for reallocation after use has completed. Resources include CPU time, memory and I/O.

As in most priority based systems, the highest-priority task that's ready to run uses the CPU; others are put on a ready list. After servicing an interrupt, the nucleus returns the CPU to the highest-priority ready task.

Managing memory space

A memory manager, built into the nucleus, handles the 8086's megabyte addressing range, insuring that memory is used efficiently. The memory manager organizes memory into a tree-structured hierarchy of pools according to each job's requirements, and provides for allocation and deallocation of memory from these pools. When the nucleus or application tasks require memory, the memory manager allocates it on the basis of segments that are multiples of 16-bytes (to 65,536 bytes) corresponding to the 8086's segmented memory architecture.

The RMX/86 nucleus provides three sets of facilities for tasks to communicate with one another. Each of the three is optimized for either data transfer, synchronization or mutual exclusion.

Mailboxes are provided to allow tasks to send data, in the form of segments, to one another. Since each seg-

ment is referenced by a description, of course, only pointers are moved rather than massive blocks of data.

Semaphores offer low overhead mechanisms for synchronization operations. For example, one task can simply set a semaphore to tell another task that an event has happened ("Analog data received").

The third communication facility offers a unique mutual exclusion facility. Regions are defined as a body of code which manipulates a shared resource. The RMX/86 nucleus insures that while one task is manipulating the resource others don't inadvertently affect it.

All intertask communication functions are accessible with simple calls to the nucleus. For example, to access a mailbox the task simply names the mailbox desired. The programmer has no need to know the internal structure of the mailbox, since all functions are accessible through easily programmed interfaces.

Fault tolerance

Another RMX/86 feature, exceptional-condition handling makes error handling simple rather than elusive. Programs can be designed to manage error conditions and take the appropriate corrective action.

First, there's an option to specify to the nucleus whether or not there should be any error handling for a particular task. A programmer can use a default handler to abort a task or he may program a specific course of action—for example, load a fresh copy of the program and try again.

The exception-condition facilities also detect programming errors such as a wrong call to the nucleus and system problems, like insufficient memory. All-in-all the OEM will find that fault tolerance is no longer just a buzzword.

Beyond the microprocessor

Most microcomputers are used with a range of peripheral devices, from serial lines to mass storage.

The RMX/86 I/O system provides the user with a very general file concept; that of files as a data sink or source. The characteristics of specific storage medium dictate the access techniques for a given file. For example, a disk file may be accessed either in sequential or random fashion, while a file accessed over a serial link (USART) must be processed serially.

Using the data sink/source concept, the user can develop application programs without worrying about the physical characteristics of the device.

Such device independence simplifies application programming and allows programs to be used with a variety of devices.

The RMX/86 I/O system supports three types of logical files:

Physical files represent the lowest interface level which retains its device-independent characteristics. Physical files provide a simple, consistent interface to all device drivers. OPEN, CLOSE, READ, WRITE,

SEEK, and special instructions perform all desired I/O operations.

Stream files provide a temporary data-transmission path between tasks. One task may write data to the stream file while another reads it. The I/O system performs the required synchronization and buffering. Stream files offer the user a simple mechanism for passing data between tasks—one that remains consistent with other file options, thus allowing the user to simulate an external device while waiting for hardware to be built.

Named files are used for the conventional data storage on mass-storage devices like floppy or hard disks for later access. However the RMX/86 I/O system goes one step further by setting up a hierarchical directory of files. This feature allows the user to organize his files to be consistent with his application.

The named-file system has another advantage: It permits file-access checking, so a user can decide which of his files he wants to protect and which to share with other users.

Each of the file options can be configured independently; so that the user may select only the features he needs—nothing more.

Furthermore, the RMX/86 I/O system is designed to allow the user to easily add his own device drivers to the system as the need arises.

The RMX/86 operating system was designed to offer the user a wide spectrum of convenient human interface functions. For example, the user of mass-storage systems is provided display directory and copy file utilities. Additionally, the OEM who needs his own interactive capability can either use the standard RMX/86 facilities or easily extend the system's human interface routines to meet his specific application requirements.

Summary

The software crisis can be overcome only by introducing a broad base of system software functionality into the market, allowing OEMs to concentrate on their area of expertise.

Intel's RMX/86 operating system takes a major stride forward in providing extendible, proven tools for OEMs to use in creating custom operating systems for their application.

The RMX/86 operating system thus provides the OEM with a powerful new system building block, enhancing the productivity of system generation. New dynamic tools like the RMX/86 operating system allow users to deliver their product into the marketplace much earlier, thereby capturing an important competitive advantage.

Modular Multitasking Executive Cuts Cost of 16-Bit OS Design

Joseph Harakal
Electronic Design, March 15, 1980

15, 1980

Joseph Harakal
Electronic

Joseph Harakal
Electronic Design,

AFN-01931A

Modular multitasking executive cuts cost of 16-bit-OS design

A modular, real-time multitasking operating system for single-board computers allows custom operating systems to be assembled largely from off-the-shelf software components. Such systems, when needed in OEM single-board μ C applications, pose problems—rarely can an OEM afford man-years of effort to develop the intimate familiarity with the hardware that's needed to design executive software. But with Intel's RMX/86 system for iSBC 86 single-board computers, he won't have to.

In addition, this second-generation, 16-bit system added error-handling, flexible command-line decode, and other advanced OS capabilities to previous options available on the older RMX/80.

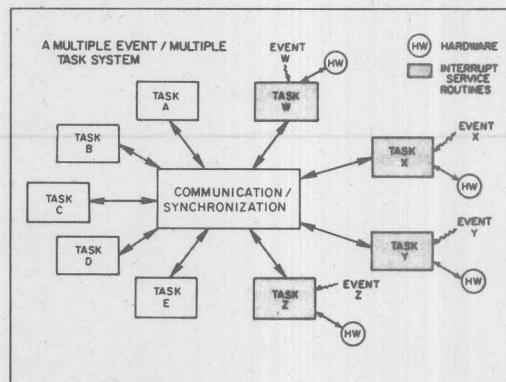
All real-time multitasking systems require executive software not only to manage the resources shared by the task programs (CPU time, memory and I/O), but also respond to interrupts and then allocate the resources according to established priorities. Normally, these functions are intermingled in an OS with higher-level system functions that often prove superfluous in single-board computer applications.

The RMX/86 OS package combines all those required executive functions in a single module, called the nucleus. Other modules in the package tailor the executive system to its application by adding higher-level OS functions such as disk-file systems. The task programs and optional modules connect to the nucleus with simple software interfaces. Users can also add their own extensions.

Since the nucleus is essentially open-ended, it can serve as the software foundation for expanding both the operating system and the variety of task programs. Although most single-board computer applications are dedicated, many do require higher-level capabilities.

The OEM way

The modular approach to system software fits in with the way most OEMs (and high-volume end users) apply single-board computers. They generally start with a minimum amount of hardware (often, just a



1. In a real-time multitasking system, task modules (A through E) can often perform their functions only after hardware-generated interrupts are serviced (highlighted). The executive in such a system provides intertask communications and synchronization.

RMX/80 vs RMX/86 features

RMX/80	RMX/86
Nuclei	Nuclei
For iSBC 80/10	One serves all iSBC boards
For iSBC 80/20	Free-space manager
For iSBC 80/30	Exceptional-conditions handler
Optional modules	Optional modules
Disk-file system	I/O system
Disk I/O	Hierarchical file system
Terminal handlers	Numbered file system
Free space manager	Internal file system
Analog I/O handlers	Physical file system
Bootstrap loader	Interfaces for custom files and I/O
Debuggers	Human interface system
Support packages	Command-line decoder
Fortran-80 run-time	
Basic-80 interpreter	
8080/8085 fundamental support	

Joseph Harakal, Software Product Manager, Intel Corp., 5200 Elam Young Pkwy., Hillsboro, OR 97123.

single board) to begin an application at low cost. Then, when users are satisfied with the original functions and are willing to buy more, the OEM adds the executive options, tasks and hardware—with as fast a turnaround as possible.

The problem with conventional "general-purpose" software systems is that they usually depend on hardware that the application may not need—for example, standard peripherals whereas a typical OEM system uses special peripherals. Modular OSs are designed to accommodate both.

What's more, a conventional system can make it difficult and/or awkward to use new peripherals or new technology, such as magnetic-bubble memory—most command-line decoders, for instance, are not accessible to the user. So, a user may discover that there is no straightforward way of adding new facilities.

Call it foundation software

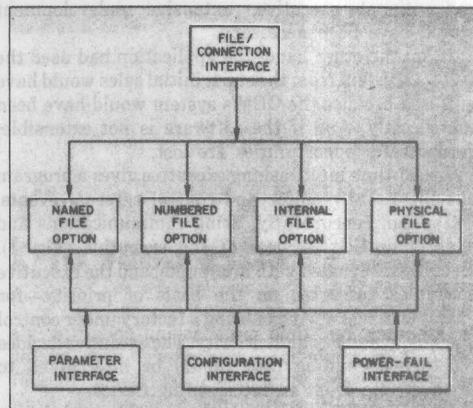
For an even closer fit with single-board computer applications, the RMX/86 modules closely parallel hardware modularity: Each computer board contains program and data memory, serial and parallel I/O, and other generally required functions in addition to the CPU. Each user's system is expandable with optional modules. Frequently used devices like disk controllers and analog I/O are available. In addition, the user can connect custom devices to his system via the Multibus architecture.

Corresponding to the hardware, systems software manages CPU, memory and I/O resources. Linked to optional modules, it can support standard iSBC devices like consoles and disk controllers. Similarly, users may add device-driver software modules for their custom peripherals. All software can reside in EPROM/ROM if mass storage is not available; otherwise, most of the system can be disk-resident. The disk-file module is suitable for such applications as data logging.

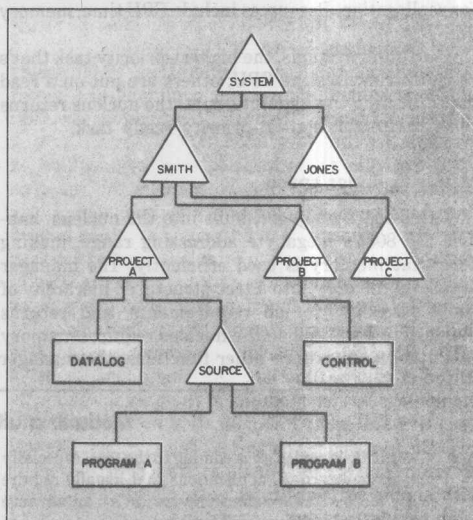
The RMX/86 is designed for configuration on an Intellec development system according to user requirements. The same system supports task-module development as well as linking and locating in both high-level and assembly languages. It also provides libraries of frequently used program functions to minimize the amount of code the system designer must write and debug.

To see how the RMX/86 works, consider a typical example. Say an OEM develops a factory heating and air-conditioning control system having a single-board computer, analog I/O, special control devices and an operator terminal. He uses the nucleus, analog-handler and terminal-handler modules, plus user tasks stored in EPROM.

But suppose the OEM's customer wants to enhance the system with, say, disk storage. He simply adds a disk controller board and uses I/O system software. The original programs could also be made disk-resident. If the original application had been based on a



2. The RMX/86 operating system treats all I/O as files—a feature that makes it easy to add new peripherals and special files.



3. A hierarchical file system eliminates the need for scanning all the files on a disk. If Smith is working on Project A, he only has to choose between the files related to his project.

conventional executive, extensive redevelopment would have been necessary.

If, on the other hand, the application had used the full I/O system from the start, initial sales would have suffered because the OEM's system would have been more costly. And if the software is not extensible, future sales opportunities are lost.

A real-time multitasking executive gives a program the means to monitor and control external events. Tasks run concurrently, using communications and synchronization services of the executive (Fig. 1). Events are signaled with interrupts, and the executive schedules resources on the basis of priority—for example, a task trying to bring a factory under control should have a very high priority. The executive decides whether a running task should be interrupted to process data from an interrupting device.

The RMX/86 nucleus makes such event-driven priority scheduling happen through resource management. It monitors system states, determines task requirements, allocates resources and gathers them for reallocation. Resources include CPU time, memory and I/O.

As in other systems, the highest-priority task that's ready to run uses the CPU; others are put on a read list. Finishing the high interrupt, the nucleus returns the CPU to the highest-priority ready task.

Managing inner space

A free-space manager, built into the nucleus, handles the 8086's megabyte addressing range, making sure that memory is used efficiently. The manager organizes memory into a tree-structured hierarchy of pools according to job requirements, and returns memory to pools. When the nucleus requires memory for a job, mailboxes or other functions, the manager

provides it. Or, when a task requests memory—for example, to input data—the manager allocates it. This is done in segments that are multiples of 16 bytes, corresponding to the 8086's segmented memory.

Tasks send data to each other through mailboxes which contain messages located in RAM. As in other systems, separate mailboxes are used for receiving and for responding.

In the RMX/86, however, mailboxes provide several options, including synchronization, mutual exclusion (which prevents one task from destroying another task's data) and communications—for example, with the outside world—through modules such as the terminal handler.

The RMX/86 nucleus also provides other means for communications. One example is semaphores—low-overhead mechanisms for synchronization operations, resource allocation and mutual exclusion that require a simple flag. For example, one task can simply set a flag to tell another task that an event has happened ("Analog data received").

All these functions are accessible with simple calls to the nucleus. For example, just two calls are needed to use the free-space manager: CREATE-SEGMENT to request memory and DELETE-SEGMENT to relinquish memory. Likewise, to obtain an mailbox, the task simply names the mailbox desired. The programmer doesn't need to know the internal structure of the executive, since the functions are accessible through easily programmed interfaces.

Errors, big and small

Another RMX/86 feature, the exceptional-condition handling, makes error handling selective rather than all or nothing. Programs can be designed to manage error conditions and take corrective action.

Modular multitasking comes on

Multitasking design is coming to the fore, especially for single-board- μ C applications that usually require a lot more software than previous μ Cs—an advance from simple foreground-background programs to techniques based on event priorities.

Although single-board μ Cs started out in the mid-1970s at the low end of the OEM performance range, they have now reached the top in performance and memory capacity. As more and more OEMs and users take advantage of that increased capability, the size of applications programs grows—and grows.

In the same period, the costs for developing software, salaries and overhead have almost doubled. Moreover, skilled programmers have become one of the industry's most limited resources. No wonder that software costs comprise up to 80% of system development costs today, and that the emphasis has shifted from in-house software design to buying off-the-shelf programs.

Because multitasking designs have to be highly modular, time-saving tools such as high-level lan-

guages, program libraries and off-the-shelf software can be used freely to help keep development, maintenance and expansion costs under control. Code written in high-level languages is a bargain today, compared to code written in assembly language: around \$2.50 a byte vs \$10 for assembly language. High-level code is not as compact, but it's far more cost-effective for the 80% of the tasks that run only about 20% of the time in typical applications.

Today, there's a growing choice of languages. Structured languages like PL/M and Pascal fit well into the "top-down" modular design technique used to divide an application into tasks. Others, like Fortran for mathematical applications and Basic for easy end-user programming, are also available.

In general, a real-time multitasking executive offers a reasonable choice for the user who has a lot of software to write, must meet special requirements, and has no time to develop a custom operating system. The RMX/86 system, with its modular design, fills the bill to save development cost.

First, there's an option to specify to the nucleus whether or not there should be any error handling for a particular task. A programmer can write his own handler either to abort a task or to program a specific course of action—for example, report exceptional conditions and continue with next instruction; load copy of module and try again; start alarm program.

The exceptional-conditions support also detects programming errors such as a wrong call to the nucleus and system problems like insufficient memory. Naturally, the RMX/86 provides all normal OS functions. System options include a terminal handler for CRT and TTY consoles device drivers for Intel's floppy and hard-disk controllers and an integrated I/O system. A subsystem of the I/O system supports tree-structured directories hierarchical named files (Fig. 2).

I/O features are vital

Most single-board computers are used with special peripheral devices, and many with other kinds of files and media. So, the I/O system is designed to make it easier to add special files, new peripherals and custom device drivers—the user need never feel locked in.

The RMX/86 I/O system provides the user with a very general file concept—as a data sink or source.

The characteristics of a specific storage medium dictate the access techniques for a given file. For example, a disk file may be accessed either in sequential or random fashion, while a file accessed over a serial link (USART) must be processed serially.

Using the data sink/source concept, the user can develop application programs without worrying about the physical device where the data will be stored. Such device independence simplifies application programming, and existing programs can be used with many devices.

The RMX/86 I/O system supports three types of files.

Physical files represent the lowest interface level to retain device-independent characteristics. They provide a simple, consistent interface to all device drivers. OPEN, CLOSE, READ, WRITE, SEEK and special instructions perform all desired I/O operations.

Stream files provide a temporary data-transmission path between tasks. One task may write data to the stream file while another reads them. The I/O system performs the required synchronization and buffering. Stream files offer the user a simple mechanism for passing data between tasks—one that remains consistent with other file options. The user can simulate an external device while waiting for hardware to be built.

Named files are used for the conventional data storage on mass-storage devices like floppy or hard

```

1      TASKB1: PROCEDURE PUBLIC;
2
2      call rgend$inittask;
2      CALL INIT;
2      do forever;
3
3          msg$token=r$receive$message(mailbox$x,
-          0FFFFH,@resp$ex,@ex$val);
3          call r$send$message(r$normal$tsout,
-          msg$token,out$resp,@ex$val);
3          msg$token= r$receive$message(out$resr
-          ,0FFFFH,@resp$ex,@ex$val);
3          call r$delete$segment(msg$token,@ex$val
-          al);
3
3          end; /* of do forever */
2      end;

```

4. RMX/86 can easily be expanded with user-coded tasks. The one shown here initializes a user program by calling the procedure INIT and helps display messages. R\$RECEIVE\$MESSAGE is a system primitive that examines the mailbox to be serviced and places the token for the first

message there (MSG\$TOKEN). Another system primitive, R\$SEND\$MESSAGE, puts the token for the message into the terminal handler's output mailbox. The primitive R\$DELETE\$SEGMENT clears the used memory and returns it to the free-memory manager.

A postgraduate system

The lessons learned since 1977 about OEM requirements for executives have fueled the evolution of the RMX/86 system. This has led to the powerful new features of the RMX/86 system.

The RMX/80 began with a nucleus for the first single-board computer (ISPC 80/10). Subsequent boards contained more memory and offered higher throughput. Nuclei for the three later boards were made interchangeable, to act as the "software bus" for transporting applications software from one board to another. The RMX/86 solution is simpler. The new nucleus is hardware-independent so it can be used on future as well as current ISPC 86 boards.

As for process-control designers have generally refused to add user-programming facilities, to assure that users do not interfere inadvertently with tasks handling critical process conditions. If a program dies during a chemical reaction, for example, a whole batch can be ruined and the processing facilities may have to be shut down.

The RMX/86 system, however, incorporates facilities for keeping programs alive. Its nucleus contains an exceptional-conditions support. Actually, a powerful error-processing subsystem, it allows single-board computers to be made fault-tolerant with no adverse impact on throughput.

disks for later access by another system. However, the RMX/86 goes one step farther by setting up a hierarchical directory of files. This feature lets the user organize his files to be consistent with his application (Fig. 3).

The named-file system has another advantage: It permits file-access checking, so a user can decide which of his files he wants to protect and which to share with other users.

Each of the file options can be configured independently; the user may select the features he needs—neither more nor less. Furthermore, the user can add his own device drivers to the I/O system.

The RMX/86 is designed to offer the user a wide spectrum of convenient functions (Fig 4). For example, the user of mass-storage systems has a display directory and copy files available. Or, the OEM who needs his own interactive capability can easily extend the system's human interface routines to meet his requirements.

As μ P applications expand, so does the need for loaders that allow parts of the applications software to reside on disks. The RMX/86 package provides a resident system loader that permits loading for either absolute or relocatable format.■

How useful?

Immediate design application

Within the next year

Not applicable

Circle No.

541

542

543

That there's an option to specify to the nucleus whether or not there should be any error handling for a particular task. A programmer can write his own handler either to abort a task or to program a specific course of action—for example, report exceptional conditions and continue with next instruction; load copy of module and try again; start alarm program. The exceptional-conditions support also detects program errors such as a wrong call to the nucleus and system problems like insufficient memory. Naturally, the RMX/86 provides all normal OS functions. System options include a terminal handler for CRT and TTY consoles; device drivers for Intel's floppy and hard-disk controllers; and an integrated I/O system. A subsystem of the I/O system supports tree-structured directories hierarchical named files (Fig. 3).

I/O features are vital

Most single-board computers are used with special peripheral devices, and many with other kinds of live and media. So, the I/O system is designed to make it easier to add special files, new peripherals and system device drivers—the user need never feel locked in. The RMX/86 I/O system provides the user with a very general file concept—as a data sink or source.

1 TASKS: PROGRAM PUBLIC
2 CALL TROUBLESHOOTING
3 CALL INIT
4 CO (OVER)
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

Faster and smaller than its predecessor, the iRMX 88 real-time modular operating system eases the transition from 8-bit to 16-bit iCS, while supporting the 8087 arithmetic chip and other peripherals.

Multitasking executive speeds 16-bit micros

May 1981

tasks execute concurrently. A task is an independent program that shares system resources and communicates with other tasks. Each task has its own stack, where the registers used by the task are saved, to shorten context-switching time.

A task communicates with other tasks via messages, which convey data and synchronization information. These messages are channeled through data structures called channels. A task can send messages to an exchange using the RSEND procedure or wait for messages at an exchange (using the RWAIT procedure). In the latter case, the task does not execute until a message is available or until a certain length of time has elapsed, specified in terms of "system time units" (minimum interval of real time recognized by the system). A task that is



1. Tasks in the iRMX 88 Executive are managed by the Executive. Transitions are governed by the Executive. The bottom of the figure shows the Executive's internal structure.

Even while it eases the transition from an 8-bit to a 16-bit world, a new operating system (OS) brings enhanced performance to the stable of modular systems software. The iRMX 88 multitasking executive cuts both interrupt latency (the delay until a random event gets a response) and context-switching time (the interval during which the processor changes from one task to another). Its nucleus is, therefore, faster—and smaller—than those of preceding operating systems.

The iRMX 88 Executive provides fundamental multitasking software and components the full-featured, multiprogramming iRMX 88 OS (Electronic Design, March 16, 1980, p. 345). Through priority-based task scheduling, it concurrently monitors and controls multiple events. It provides real-time clock control, manages interrupts, and dispatches tasks. As an upgraded version of the iRMX 88 Executive, iRMX 88 employs the same OS and most of the same hardware. The new OS offers field-proven and reliable multitasking.

The Executive will run with iAPX 88 or iAPX 86-based boards and supports Intel's iSBX 86/128, iSBX 86/40, and iSBX 86/05 single-board computers. It is fully configurable—not only procedure-by-procedure, but also for all hardware addresses and interrupt levels—it also can be used with custom-designed boards that contain an iAPX 88 or iAPX 86 processor, an 8255A programmable-interrupt controller, and 8257 programmable-interval timer, an optional 8251A programmable-communications interface, and perhaps most important—an 8087 numeric-data processor.

In a multitasking system, the iRMX 88 executive

Jess M. Irwin
Electronic Design

Jess M. Irwin, Project Leader,
Intel Corp.,
2525 N. Elgin Young Pkwy., Hillsboro, OR 97124

Faster and smaller than its predecessor, the iRMX 88 real-time modular operating system eases the transition from 8-bit to 16-bit μ Cs, while supporting the 8087 arithmetic chip and other peripherals.

Multitasking executive speeds 16-bit micros

1881 vsM

Even while it eases the transition from an 8-bit to a 16-bit world, a new operating system (OS) brings thoroughbred performance to the stable of modular systems software. The iRMX 88 multitasking executive cuts both interrupt latency (the delay until a random event gets a response) and context-switching time (the interval during which the processor changes from one task to another). Its nucleus is, therefore, faster—and smaller—than those of preceding operating systems.

The iRMX 88 Executive provides fundamental multitasking software and complements the full-featured, multiprogramming iRMX 86 OS (ELECTRONIC DESIGN, March 15, 1980, p. 245). Through priority-based task scheduling, it concurrently monitors and controls multiple events. It provides real-time clock control, manages interrupts, and dispatches tasks. As an upgraded version of the 8-bit iRMX 80 Executive, iRMX 88 employs the same concepts and most of the same modules. Thus, the new OS offers field-proven and reliable interfaces.

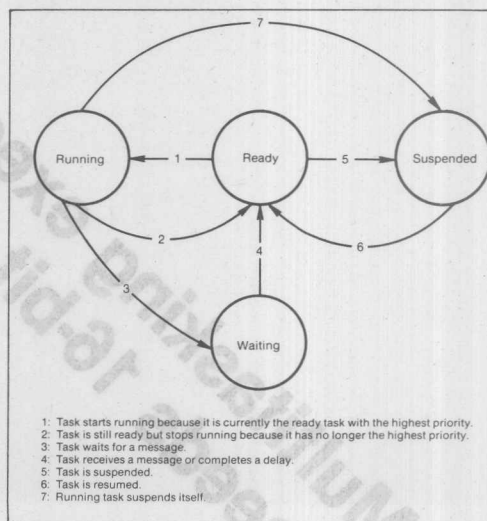
Tasks share resources

The Executive will run with iAPX 88 or iAPX 86-based boards and supports Intel's iSBC 86/12A, iSBC 88/40, and iSBC 86/05 single-board computers. Since it is fully configurable—not only procedure-by-procedure, but also for all hardware addresses and interrupt levels—it also can be used with custom-designed boards that contain an iAPX 86 or iAPX 88 processor, an 8259A programmable-interrupt controller, and 8253 programmable-interval timer, an optional 8251A programmable-communications interface, and—perhaps most important—an 8087 numeric-data processor.

In a multitasking system like the iRMX 88, several

tasks execute concurrently. A task is an independent program that shares system resources and communicates with other tasks. Each task has its own stack, where the registers used by the task are saved to shorten context-switching time.

A task communicates with other tasks via messages, which convey data and synchronization information. These messages are channeled through data structures called exchanges. A task can send messages to an exchange (using the RQSEND procedure) or wait for messages at an exchange (using the RQWAIT procedure). In the latter case, the task does not execute until a message is available or until a certain length of time has elapsed, specified in terms of "system time units" (minimum interval of real time recognized by the system). A task that is



1. Tasks in the iRMX 88 Executive occupy one of four states. Transitions are governed by the seven occurrences listed at the bottom of the figure.

Jess M. Irwin, Project Leader
 Intel Corp.
 5200 N.E. Elam Young Pkwy, Hillsboro, OR 97213

IRMX 88 Executive

not willing to wait may accept any message from an exchange (using the RQACPT procedure) if one is available; otherwise, the task remains without a message.

Each task exists in one of four states: running, ready, waiting, and suspended (Fig. 1). A task is running if the processor is executing instructions on its behalf. It is considered ready if it is either running or able to run. A task is waiting if it has requested a message but has not received it. Finally, a task is suspended if some other task has specifically requested that it not be permitted to run.

Each task has a permanent priority, which indicates its importance relative to other tasks in the system and to interrupts from peripherals. Priorities range from 0 to 255, with 0 being the highest priority. The "idle task" executes at priority 255 and prevents any other task at that priority from running. The ready task with the highest priority is the running task. The software priorities correspond to eight hardware interrupts (Fig. 2). When the running task has a high priority, interrupt levels associated with equal or lesser priorities are masked off.

Interrupts provide real-time access

An interrupt invokes an interrupt-service routine, which either requests an end of interrupt (using the

RQENDI procedure) or requests that a message be sent (using the RQISND procedure) to an interrupt exchange associated with the active interrupt's level (Fig. 2). In the latter case, an interrupt task waiting at the interrupt exchange would complete the interrupt servicing. The system can enable interrupts (via the RQELVL procedure) and disable them (via the RQDLVL procedure) and can set the interrupt vector associated with a given priority level (using the RQSETV procedure).

The iRMX 88 treats interrupts as messages. When an interrupt occurs, the Executive creates a special message and sends it to the exchange associated with that particular interrupt. Should a previous interrupt message be at the exchange when the interrupt occurs, the message is changed to indicate a missed interrupt. Any task awaiting the interrupt exchange receives the interrupt message and is readied for execution. If the task has a priority corresponding to the interrupt level, it will become the running task. The task that was running when the interrupt occurred is still ready to run, but is pre-empted. When the running task relinquishes control by waiting at an exchange (possibly for another interrupt), suspending itself (via the RQSUSP procedure), or deleting itself (via the RQDTSK procedure), the iRMX 88 dispatches the next ready task.

Interrupt-processing tasks are simply tasks that wait at interrupt exchanges. They do not need to save the interrupted task's state or control the interrupt hardware. A task may simulate an interrupt by sending a message to an interrupt exchange (using the RQSEND procedure).

Because the iRMX 88 imposes some software overhead, a direct interrupt-servicing facility is provided for those cases where interrupt response time is critical. The interrupt vector is set with a pointer to an interrupt-service routine (using the RQSETV procedure); then all interrupts occurring at that level are handled by the interrupt-service routine, which is also responsible for all interrupt-logic control and state-saving. To complete the interrupt, the routine must use the RQISND or RQENDI procedure.

Creating tasks and exchanges

The iRMX 88 supports the dynamic creation of tasks and exchanges, as well as their deletion when they are no longer needed. Creating a task (with the RQCTSK procedure) places the new task on the system's list of ready tasks, according to its priority. The creation of an exchange (using the RQCXCH procedure) results in the initialization of an "empty" exchange—one that has no waiting messages or tasks. The deletion of a task (via the RQDTSK procedure) results in its removal from all system lists. The deletion of an exchange (using the RQDXCH

Hardware	Software
	Level 0 Highest priority
Level 0	16
1	17
	32
	33
2	48
	49
3	64
	65
4	80
	81
5	96
	97
6	112
	113
7	128
	129
	255 Lowest priority

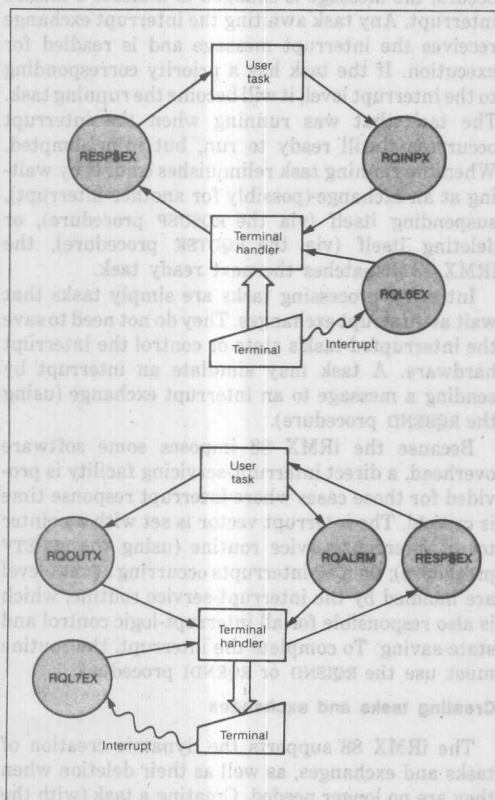
2. Of 256 interrupt levels, the top 129 are associated with hardware. The remainder can be assigned to software tasks.

procedure) is only possible if no messages or tasks are waiting at the exchange.

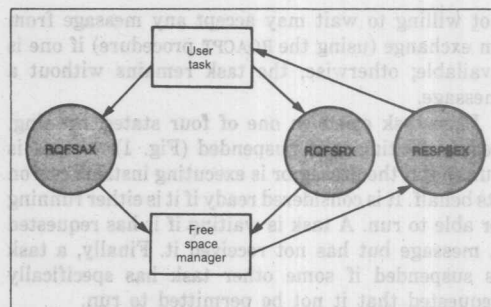
Suspension of a task (using the RQSUSP procedure) places the task on a list of suspended tasks and removes it from the list of ready tasks; to return the task to the ready state, the RQRESM procedure must be used.

A coprocessor for number crunching

Unlike previous operating systems, the iRMX-88 can support the 8087 numeric-data processor (NDP) in a way that is invisible to the user. When an NDP task is created, the iRMX 88 initializes and maintains the state of the 8087 in a save area. Should an



3. Interaction between the terminal and a user task involves several iRMX 88 exchanges (gray). Some serve for inputs from the keyboard (top); others, for outputs to the CRT (bottom).



4. When a user task needs a block of RAM, exchange RQFSAX passes the request to the free-space manager, which sends an acknowledgment via RQSPBX. The RQFSRX exchange serves to release the block.

DESTINATION:	POSITION:
1.0000 METERS	X = 1.2000 METERS
4.0000 METERS	Y = 3.4000 METERS
5.0000 METERS	Z = 2.1000 METERS
MAXIMUM VELOCITY:	VELOCITY:
0.0100 METERS/SECOND	DX = -0.0050 METERS/SECOND
0.1000 METERS/SECOND	DY = 0.0800 METERS/SECOND
0.0500 METERS/SECOND	DZ = 0.0500 METERS/SECOND

5. The arm of an industrial robot can move in three directions: X, Y, and Z. The console display reports current position and velocity (left), as well as the limiting values (right).

unmasked error occur in the 8087, an interrupt is generated on a configurable level. Using the default interrupt-service routine, the iRMX 88 sends an interrupt message to the interrupt exchange; the user can then supply his own error handler. The error task must specify itself as a non-NDP task or the system will deadlock. But the error interrupt, if used, must not be masked off.

The Executive includes a terminal handler which establishes real-time asynchronous serial communication between an operator's terminal and the tasks running under the executive (Fig. 3). The terminal handler provides a simple operator interface, which includes line-editing features, a 64-byte type-ahead buffer, and control over the output.

Since all port addresses and interrupt levels for the terminal handler are configurable, it supports any component configuration using an 8251A programmable-communications interface with an 8253 programmable-interval timer for baud-rate generation. A task interfaces with the terminal handler through two exchanges, where messages are sent to initiate read and write requests; they are processed on a FIFO basis. Characters are read and written in response to interrupts from the 8251 programmable-communications interface. Each request results in the transmission of one line to or from the terminal. Read requests are sent to the RQINPX exchange, and write requests to the RQOUTX

IRMX 88 Executive

exchange. When a request has been serviced, its message is sent to the specified response exchange.

The iRMX 88 also contains a free-space manager, which maintains a pool of free RAM memory (Fig. 4). Tasks request blocks of memory from the pool and return blocks to the pool. The request is made by sending a message to the manager's allocation exchange, RQFSAX, specifying the length of the needed block. If sufficient contiguous free RAM is available, the requesting task receives a message acknowledging the request and supplying the address of the block in the form of a message. Otherwise, the requesting task receives a message indicating how much free RAM can be allocated. The requesting task may then ask for a smaller block. In either case, the response from the free-space manager is returned in the same message that was used for the request. If the task must have the block of memory, it makes an unconditional request: the

```

title 'display algorithm')
display:
DO;
$include(%f1:ied.llt)
$include(%f1:rqiend.ext)
$include(%f1:rqiend1.ext)
$include(%f1:rqiwait.ext)
$include(%f1:rasetv.ext)
$include(%f1:rqiavl.ext)

DECLARE display$ied int$exchange$descriptor PUBLIC;

/* display interrupt service routine */
display$ir: PROCEDURE INTERRUPT 63;
/* check for proper interrupt */
OUTPUT(rq$S259a)=0bh;
IF NOT ((INPUT(rq$S259a) AND 80h) = 0) THEN
DO;
/* output next character */
OUTPUT(rq$st$S251) = output$buffer(output$index);
output$index = output$index + 1;
output$count = output$count + 1;
IF output$count = 0 THEN
CALL rq$insd(.display$exchange);
ELSE
CALL rq$sendi(.display$exchange);
END display$ir;

display: PROCEDURE;
DECLARE message$address ADDRESS;
CALL rq$setv(.display$ir,display$level);
DO FOREVER;
/* copy monitor and control information into the display
buffer with the template of the display */
.
.
.
output$index = 1;
output$count = LENGTH(output$buffer);
/* enable the display interrupt and wait until the
buffer has been output */
CALL rq$avl(.display$ied);
message$address = rq$wait(.display$ied,0);
END display;
```

6. The PROCEDURE DISPLAY (bottom half) illustrates how the IRMX 88 operating system deals with random events. Here, interrupt level 63 has been chosen by the user.

free-space manager then lets the requesting task wait until sufficient memory is available. When a block of RAM is no longer needed, it may be sent to the reclamation exchange (RQSRX) where it is merged into the free-space pool.

The building-block approach

The iRMX 88 Executive is completely configurable. The minimum "nucleus" consists of the following procedures: RQSTRT (initialization), RQCTSK (task-creation), RQCXCH (exchange-creation), RQSEND, and RQWAIT. Any of the remaining procedures are automatically included when they are referenced. Interrupt support and the system clock are separately configurable. The port addresses used for the 8259A programmable-interrupt controller, the 8251A programmable-communications interface, and the associated 8253 programmable-interval timer are configurable to any iAPX 86 or iAPX 88 port address. The transmission rate of the terminal handler is configurable from 150 to 19,200 baud. The interrupt levels for the system-clock and terminal-handler interrupts are configurable to any level of the 8259A programmable-interrupt controller. Any unused level is configured to its default interrupt-service routines. The base of the interrupt vector used by the 8259A programmable-interrupt controller can be selected from interrupt levels 8 to 248. The smallest system time unit is 1 ms.

An "interactive configuration utility" (ICU 88) simplifies the configuration process. This utility, which executes on the Intellec microcomputer-development system, asks the user a series of questions, allowing him to specify his own board configuration or to use a default that corresponds to the iSBC 86/12A, iSBC 86/05, or iSBC 88/40 boards.

To transfer an application from the iRMX 80 to the iRMX 88 Executive, the user must change the parameter for the create-task (RQCTSK) procedure from a 16-bit address to a 32-bit pointer. The description of the task can then be placed in random-access or read-only memory. In the iRMX 88, the definition of task has been expanded to incorporate a flag that indicates whether or not the 8087 numeric-data processor will be used. If that flag is set, an additional 94 bytes must be reserved for the task. Furthermore, the first parameter of the set-vector (RQSETV) procedure must be changed to a 32-bit pointer.

A robot demonstrates implementation

The iRMX 88 Executive demonstrates its value in a typical application: controlling an industrial robot's arm. The position of the arm in each of three dimensions is measured by an analog voltage on three input lines. The speed and direction of movement in each dimension is controlled with an analog

```

input: PROCEDURE;
/* initialize */
DO FOREVER;
DO CASE field + 1;
/* next x position */
DO;
/* convert the character string in the input buffer
into a real number */
CALL rq$send(.control$x$ed,
.x$position$message);
END;
END CASE;
END input;
END input;

```

7. An endless loop is a typical method for capturing input.
The skeleton code for PROCEDURE INPUT shows the interface with the executive via a call to an OS procedure.

```

$title ('monitor algorithm')
monitor:
DO;
/* monitor x task */
monitor$x: PROCEDURE;
DECLARE message$address ADDRESS;
CALL rq$cxch(.waiting$ed);
DO FOREVER;
message$address = rq$wait(.waiting$ed, 16);
OUTPUT(ac$command) = x$inputs$ed;
message$address = rq$acpt(.waiting$ed);
x$control$input = SHR( INPUT(ac$high), 4);
x$monitor$position = x$control$input * x$input$scale;
x$rate = ( x$position - x$monitor$position ) * 10;
x$position = x$monitor$position;
CALL rq$send(.x$action$ed, x$position$message);
END;
END monitor$x;
END monitor;

```

8. PROCEDURE MONITOR \$X is a task that has to run every 16 ms.
The IRMX 88 procedure RQWAIT accepts the desired time delay as an argument.

```

$title ('control algorithm')
control:
DO;
/* control x task */
control$x: PROCEDURE;
DECLARE message$address ADDRESS;
DO FOREVER;
message$address = rq$wait(.x$action$ed, 0);
x$control$current = max$x$rate *
( next$x$position - x$position ) / ABS( next$x$position
- last$x$position );
x$control$output = SHL( FIX (
x$control$current * x$scale ), 4 ) + x$output$select;
OUTPUT(ac$high) = HIGH(x$control$output);
OUTPUT(ac$low) = LOW(x$control$output);
END;
END control$x;
END control;

```

9. This control algorithm sends data to the robot arm's X-axis through a d-a converter. The task remains dormant until the INPUT procedure from Fig. 7 makes it up.

current on three output lines.

The hardware consists of an iSBC 88/40 board with an iSBC 337 high-speed math multimodule, an iSBX 328 analog-output multimodule, and a CRT terminal with an RS-232 serial interface. The position of the arm in any dimension is read from the a-d converter on the iSBC 88/40; the converter is multiplexed over the three analog inputs. The d-a converter on the iSBX 328 analog-output multimodule is multiplexed to control the current to three stepper motors, which in turn control the arm's motion. A display (Fig. 5) is maintained on the CRT terminal and gives the robot's current position in three dimensions, the rate of change in each direction, and updatable fields for the desired position, as well as the maximum rate of change in each direction.

The display algorithm (Fig. 6) continuously updates the display from the control and monitor data bases. The monitor and control data bases update the display buffer. The display task initializes the display index and display count, enables the display interrupt, and waits at the display-interrupt exchange. The display-interrupt service routine responds to display interrupts by outputting the next character from the display buffer. When the last character in the display buffer has been output, a message is sent to the interrupt exchange; otherwise an end-of-interrupt is sent.

The input algorithm (Fig. 7) accepts digits as they are typed on the CRT terminal and enters them into the control field being updated. The fields on the right-hand side of the display are updated, starting with the top field and progressing to the bottom. Entry of a carriage return causes the values in the updated field to update the control-data base; the input algorithm then proceeds to the next field. Backspace discards the last digit typed. All other characters are ignored, and the user cannot type outside the defined fields. The input-interrupt routine receives a character with each interrupt and copies it into the display and input buffers.

The monitor algorithm (Fig. 8) samples the position of the robot's arm 60 times per second. It updates the monitor data base and then signals the control program through the appropriate exchange.

The control algorithm (Fig. 9) waits at the proper exchanges for a message from the input or the monitor algorithms. It then outputs a control current to the d-a converter based on the relative position of the robot's arm. □

Choosing a VLSI-compatible system

PETER PALM, Intel Corp.

November, 1981

Two types of systems are either end-user-oriented (execution) systems or OEM (development) systems. But such a categorization is inadequate because many end users are concerned with software development and many OEMs are concerned with software execution.

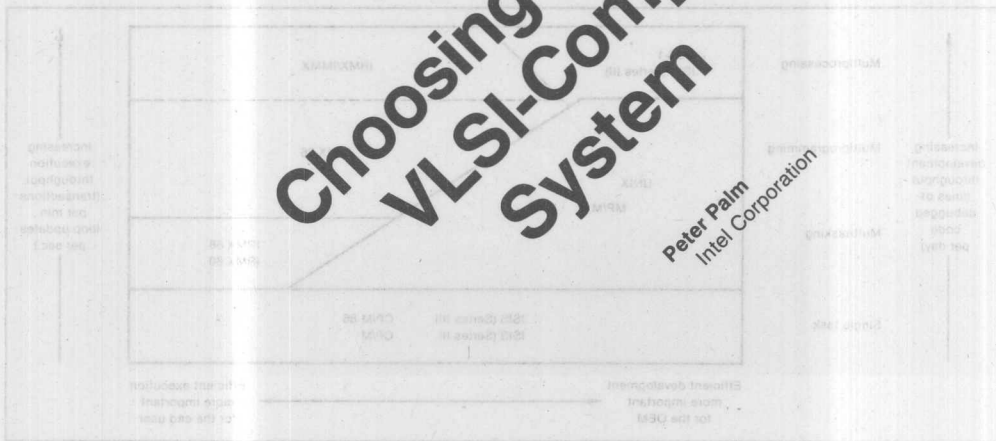
Development versus execution-oriented

In the 8-bit to 16-bit world, CP/M and Intel's iAPX (development) operating systems are good examples of products targeted at efficient software development. In the 16-bit world, the segment development in the iAPX operating system and iAPX (Series III) and the UNIX operating system and derivatives, such as XENIX, fill the development need. Software developed with one of these operating systems is often executed on another machine running an execution-oriented operating system. As more development throughput is required, such as compiling times per hour, users move from single-terminal single-task systems (such as CP/M) to multitasking systems (such as UNIX or XENIX) and to multiprocessor development systems (such as Intel's NDS-1).

By integrating more functions into silicon, very large-scale integrated (VLSI) circuitry lowers the cost of systems and improves their performance and ease of use. The advent of VLSI raises a question: which operating systems enable a user, especially an OEM, to take advantage most quickly of these hardware innovations? This question can be answered by examining the characteristics of available operating systems in the light of VLSI advances.

An operating system's intended application is a key consideration in its selection. No operating system is ideal for every application and customer. For this reason, designers tend to optimize operating systems for specific types of applications.

Microcomputer operating systems fall into two categories: efficient development of new software and those for efficient execution of existing software. Development-oriented systems tend to sacrifice performance to ease use, while execution-oriented systems make the opposite trade-off. It is tempting to



OPERATING SYSTEMS

Choosing a VLSI-compatible system

PETER PALM, Intel Corp.

*VLSI offers expanded modularity for operating systems;
the intended application will determine the most appropriate system*

By integrating more functions into silicon, very large-scale integrated (VLSI) circuitry lowers the cost of μ cs and improves their performance and ease of use. But the advent of VLSI raises a question: which μ c operating systems enable a user, especially an OEM, to take advantage most quickly of these hardware improvements? This question can be answered by examining the characteristics of available μ c operating systems in the light of VLSI advances.

An operating system's intended application is a key consideration in its selection. No operating system is ideal for every application and customer. For this reason, designers tend to optimize operating systems for specific types of applications.

Microcomputer operating systems fall into two categories: systems optimized for efficient development of new software and those optimized for efficient execution of existing software (Fig. 1). Development-oriented systems tend to sacrifice performance to ease of use, while execution-oriented systems make the opposite trade-off. It is tempting to characterize the

two types of systems as either end-user-oriented (execution) systems or OEM (development) systems. But such a categorization is inadequate because many end users are concerned with software development, and many OEMs are concerned with software execution.

Development- versus execution-oriented

In the 8-bit μ c world, CP/M and Intel's ISIS (Intellec development system) operating systems are good examples of products targeted at efficient software development. In the fast-growing 16-bit μ c segment, ISIS (Series III) and the UNIX operating system and derivatives, such as XENIX, fill the development need. Software developed with one of these operating systems is often executed on another machine running an execution-oriented operating system. As more development throughput is required, such as compilations per hour, users move from single-terminal, single-task systems (such as CP/M) to multiterminal systems (such as UNIX or MP/M) and to multiprocessor development systems (such as Intel's NDS-1).

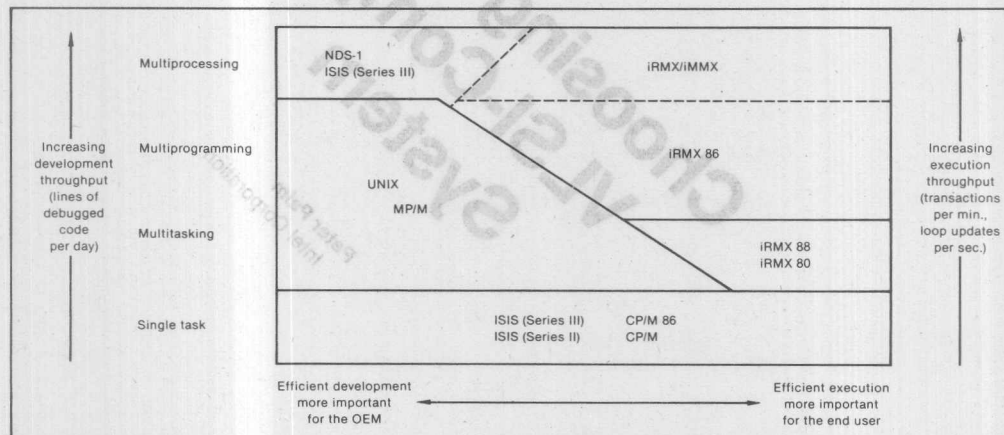


Fig. 1. μ C operating systems are design-optimized for efficient execution for end users (right) or for efficient program development for OEMs (left). They range from single-task, single-terminal systems (bottom) to multiprocessing systems (top).

An operating system's intended application is key in its selection.

Most μ c systems are execution-oriented and are often most critical for OEMs who must stay competitive in price and performance. Rapid program development is usually secondary to efficient software execution. An OEM can gain a significant competitive advantage by using an operating system that provides faster execution times, less expensive investment, ease of use and the ability to be upgraded.

Operating system vendors, including Intel, have designed a range of products for different needs. CP/M, for instance, is a logical candidate for 8-bit μ c applications, in which only single-task execution is required. For 16-bit applications, multiprogramming-type operating systems, such as MP/M, iRMX 86 and iRMX 88, are more appropriate because they take advantage of the 16-bit processor's power. An OEM for whom real-time execution is important might consider iRMX 86. If background program development is crucial, MP/M might be a better choice. For highest performance, users should consider multiprocessing operating systems, such as iRMX/MMX800.

Other selection criteria

Although the intended application is paramount, other considerations are important in selecting an operating system. The overriding factor in light of current trends is probably the ability of a system to keep pace with the impact of VLSI on μ c performance and cost. This consideration entails several selection criteria, including transferability, multiprocessing architecture, configurability and interfacing to modules.

First, an operating system should be easy to transfer—at least in part—into VLSI silicon. Putting an

operating system into EPROM, for example, can improve speed and lower cost. Vendors should state which operating system functions will be integrated into silicon, and when.

An operating system should provide support for multiprocessing, and VLSI has made multiple- μ c systems practical. OEMs should look for operating systems with multiprocessor architectures or other features that ease the move to multiprocessing. Such systems typically include fast context switching, task-to-task communication, synchronization and memory message passing features. For example, the new iMMX800 Multibus message-exchange software allows 8- and 16-bit, master-to-master and master-to-intelligent-slave single-board computers to multiprocess loosely on the Multibus multiprocessor bus.

Modularity is another important criterion in selecting an operating system. The iRMX 86, for example, consists of layers of modules that can easily be moved into silicon as required (Fig. 2). This modular design has enabled Intel to develop a new component dubbed 80130 Operating System Firmware (iOSF) that integrates a timer, an interrupt controller and the iRMX 86 kernel.

An operating system should include standard interfaces to modules. For example, iRMX 86 uses a standard object-oriented format for interfaces to jobs, tasks and message primitives. At a higher layer, iRMX 86 offers standard device-independent interfaces to drivers for the new 8089/8272-based Winchester- and floppy-disk controllers and other device controllers. The interface itself eventually will move into silicon. Only standard interfaces will allow this to occur.

An operating system should also provide industry-standard interfaces to popular program-development languages, such as Intel's universal development interface (UDI) and universal run-time interface (URI). Intel's new UDI/URI-compatible languages, FORTRAN 86, Pascal 86, PLM/86 and ASM 86, can run on any UDI/URI-compatible operating system.

Operating systems should either support or should soon support the leading local-area networks, such as Ethernet, and global-area networks, such as X.25 2780/3780. In November, iRMX 86 will provide the first high-level support for Ethernet, the tri-corporate Digital Equipment Corp., Intel Corp. and Xerox Corp. local-area network standard. Prestigious firms committing to Ethernet include Hewlett-Packard Co., Siemens Corp., Nixdorf Computer Corp., Olivetti Corp. and Zilog Corp. Intel will provide high-level, data-link-layer interfaces to an Ethernet controller (iSBC 550) on the standard Multibus, via the new Multibus interprocessor protocol (MIP). Ethernet will be supported in iRMS 86 via iMMX. These are all standard modules with standard interfaces.

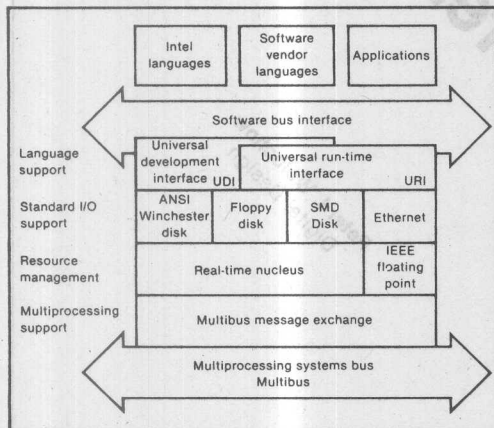


Fig. 2. Layered operating systems with standard interfaces, standard modules and configurability are key elements in designing μ c operating systems to take maximum advantage of lower cost, higher performance VLSI.

Peter Palm is systems and software product marketing manager, OEM Microcomputer Systems Operation, Intel Corp., Hillsboro, Ore.

SOFTWARE

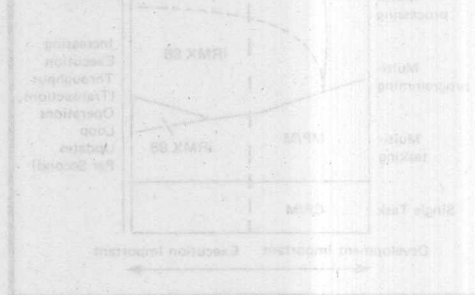
development-oriented operating systems are developed on a maintenance or microcomputer with a cross-compiler. The other alternative is to develop the software on a microcomputer using a development-oriented operating system that has tailored human interface more to the needs of the programmer than user of the end product.

In addition to the standard support of an operating system — simple human interface, initialization, and resource management — operating systems for efficient program execution have a number of other objectives as well. These include real-time operation, multiprocessing, multi-tasking, multiprocessing, and efficient scheduling and priority determination.

Microcomputer Operating System Trends

advances in VLSI and other technologies force the development of advanced operating systems

As the use of μ Cs expands, demand for high-level languages and human machine methods of interfacing will expand in 1982 and beyond. One major component of the expanding demand concerns μ C operating systems.



by Peter I. Wolochow

An operating system performs resource management and human-to-machine translating functions. Technically, it is just another computer program — a series of instructions that tell the machine what to do under a variety of conditions. Major operating system functions include management of memory, I/O peripherals, and the central processor.

When computers were first developed, the programs (or instructions) were entered into the machine each time a particular job was started. Only with the ability to store a program in the computer's memory, over 30 years ago, did the concept of computers as we know them today truly emerge. The ability to store a program meant that a computer could perform repetitive tasks while the operator had only to enter information upon which calculations were performed.

Once it became possible to store programs, the development of operating systems began. Operating systems were stored in the computer's memory, and provided the user with a bank of stored computer instructions that could be used with a number of different application programs.

Over the years, operating systems have evolved to the point where they have three main purposes. First, they provide clear, consistent, and easily understood guidelines to users concerning how the machine works. A sub-objective is to provide an easier, more "friendly" human interface to the μ C. Second, they perform initialization and start-up functions "automatically" so that — to the user — the machine is ready to perform its basic functions. This initialization function originally included only initial start-up, but now often includes methods to recover from errors in both hardware and software. Third, they provide efficient machine and storage resource management so that different users or programs can

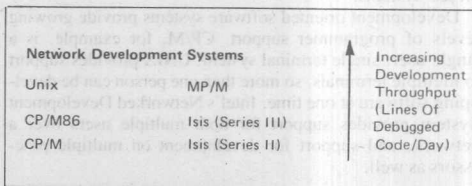


Figure 1: These microcomputer operating systems are primarily for efficient program development.

Peter I. Wolochow is with the OEM Microcomputer Systems Div. of Intel Corp., Hillsboro, OR.

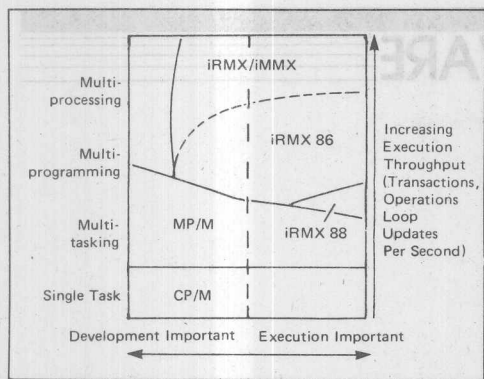


Figure 2: These microcomputer operating systems are primarily for efficient execution.

share the same resources, (e.g., memory, peripherals, I/O ports, a floating point program or the processor itself).

Although virtually all μ C operating systems provide users with methods to accomplish these three main purposes, certain operating systems — designed for specific types of applications — have gone far beyond the three common objectives.

One method of viewing μ C operating systems has to do with the primary purpose to which the μ C is directed. Two major distinctions exist. On one hand, some μ C operating systems are used primarily to develop new software systems. In this case, the operating system often includes only features and routines that are useful to persons writing and compiling software programs.

On the other hand, other μ C operating systems are directed primarily at efficient execution of software programs for various applications. In this case, the operating system often includes routines and abilities targeted to fast, easy program execution.

software development OS

Within the current world of μ C operating systems, examples of those oriented to efficient software development include such products as UNIX (developed by Bell Laboratories) and its related XENIX from MicroSoft, CP/M and CP/M-86 from Digital Research, and ISIS and Networked Development Systems from Intel. CP/M and ISIS-Series II are examples of operating systems designed for the technology and power of 8 bit μ C systems. UNIX was designed to match the 16 bit minicomputer, (e.g., DEC PDP-11) and now is being ported (e.g., XENIX) to the newer, more powerful 16 bit μ C systems.

Development oriented software systems provide growing levels of programmer support. CP/M, for example, is a single user, single terminal system. UNIX provides support to multiple terminals, so more than one person can be developing software at one time. Intel's Networked Development System provides support for both multiple users over a network, and support for development on multiple processors as well.

execution oriented OS

The second major category of operating systems comprises those primarily targeted to meet the needs of efficient execution of application software already written, developed, and tested. This category includes the largest number of systems,

and is often critical to effective utilization of μ Cs in applications such as industrial control, communications, transaction and word processing, interactive graphics, and simulation.

Often, execution programs working in conjunction with execution-oriented operating systems are developed on a mainframe or minicomputer, with a cross-compiler. The other alternative is to develop the software on a μ C using a development oriented operating system that has tailored its human interface more to the trained programmer than users of the end product.

In addition to the standard purposes of an operating system — simple human interfaces, initialization, and resource management — operating systems for efficient program execution have a number of other objectives as well. These include real-time operation, multiprogramming, multitasking, multiprocessing, and effective scheduling and priority determination.

real-time operation

Many real-time operations are dedicated to specific applications, such as controlling a machine or series of machines. As such, they often run the same software programs over and over again, depending on inputs received from monitoring and measuring devices attached to the machine they control.

The primary characteristic of such applications is that the data or input that causes the μ C to act is not regular, and does not occur at a particular rate. As a result, the μ C program and operating system must be able to handle inputs as they occur, and to monitor or control activities based on these real-time inputs.

multiprogramming

This refers to the ability of an operating system to support several independent applications executing concurrently. If a μ C, for example, is used to provide an integrated business office system, the software might be divided into a number of separate applications. One might focus on WP, another to manage the printer, and perhaps another to manage an inter-office electronic mail service. Each of these applications would involve a number of tasks devoted to different parts of the system.

An operating system that supports multiprogramming allows this division, and consequently makes it easier to develop the different applications separately, and insure that they do not interfere with each other. This is accomplished by establishing a separate environment for each application. These environments provide the basic appearance of a series of individual machines, while sharing the resources of only one. A multiprogramming operating system must manage this division, keep track of the tasks and requirements of each application, and ensure that each task is given the correct priority.

multitasking

Multitasking refers to the ability of an operating system to effectively manage several tasks, of one computer program, that are operating simultaneously. Multitasking is an important sub-set of multiprogramming, wherein several programs are concurrently being executed. Many applications require that one applications program involve different tasks. Often, these tasks must operate based on data developed in other tasks, and hence a form of task-to-task communication is required. Generally, operating systems designed for efficient program execution require some form of "executive" to manage tasks, priorities, and intertask communication. As a result, common resources such as μ P memory and I/O de-

vices can be shared by multiple tasks and can be kept as busy as possible, adding to overall system efficiency.

multiprocessing

Multiprocessing refers to the ability of an operating system to support multiple processors. In certain applications, the demands of the applications exceed the capacity of a single processor or μ C. As a result, more processors may be added. When this occurs, the role of the operating system is to efficiently allocate different processing requirements to the various processors, to keep track of which jobs have been sent where, and to assure that the total system resources are effectively utilized. Multiprocessing is becoming more attractive as a way to expand the functions of particular μ C applications without the need to entirely rewrite a program.

As μ Cs become less costly compared to software development and maintenance, many systems will use multiple processors, and operating systems will be required to efficiently manage multiprocessing configurations. A means for linkage of multiple processors and operating systems is Intel's iMMX 800 (Multibus Message Exchange) software and iSBC 550 Ethernet communications controller. They support the needs of local area network applications such as office automation, distributed data processing, factory data collection, research data collection, intelligent terminal and other EDP-related applications.

scheduling and priority determination

As μ C operating systems are required to manage even more complex functions — such as multiple programs, multiple

Processor Management:	iRMX 86	UNIX	CP/M	RSX-11M	RT-11	MTOS-86
Scheduling	Realtime	Multiprogram	Batch	Realtime	Realtime	Realtime
Multitasking	Yes (64K)	No	No	Yes (64K)	No	Yes (4K)
Priority Levels	255	None	None	250	None	255
Multiprogramming	Yes (64K)	Yes (64K)	No	Yes (64K)	Foreground / Background	No
Multiprocessor	iMMX	No	No	No	No	Yes (16)
Multiuser	Release 5	Yes	No	Yes (4)	No	No
Interrupt Management	Yes	No	No	Yes	Yes	Yes
Error Management	4 levels	Yes	No	Yes	Yes	No
Powerfail Protect	No	No	No	Yes	No	No
Memory Management:	iRMX 86	UNIX	CP/M	RSX-11M	RT-11	MTOS-86
Dynamic	Yes (1MB)	Yes (64K)	No	Optional	No	Yes (64K)
# of Memory Pools	64K	One	One	8	One	32
Memory Resident	Yes	No	Yes	Yes	No	Yes
Application Loader	Yes	Yes	Yes	Yes (11S: No)	Yes (RT ² : No)	No
Bootstrap Loader	Yes	Yes	No	Yes	Yes	No
(P) ROM'able	Yes	No	No	No	No	Yes
Device Management:	iRMX 86	UNIX	CP/M	RSX-11M	RT-11	MTOS-86
Concurrent I/O	Yes	Yes	No	Yes	Yes	Yes
I/O Buffering	Yes	Yes	No	Yes	Yes	No
Reentrant I/O	Yes	Yes	No	Yes	Yes	Yes
Asynchronous I/O	Yes	No	No	Yes	Yes	Yes
Synchronous I/O	Yes	Yes	Yes	Yes	Yes	No
Device Independent I/O	Yes	Yes	Yes	Yes	Yes	Yes
Max. # of Drivers	64K		256	256	16	256
Data Management:	iRMX 86	UNIX	CP/M	RSX-11M	RT-11	MTOS-86
File Support:						
1. Sequential	Yes	Yes	Yes	Yes	Yes	Yes
2. Indexed Seq.	No	Yes	No	Yes	No	No
3. Direct Access	Yes	Yes	Yes	Yes	Yes	Yes
Swapping	No	Yes	No	Yes	No	No
Overlays	Yes	Yes	Yes	Yes	Yes	No
Hierarchical Directories	Yes	Yes	No	No	No	No
Stream Files	Yes	Yes	No	No	No	No
Mailboxes	Yes	No	MP/M	No	No	No
Critical Regions	Yes	Yes	No	No	No	Yes
Host System For Development	Yes (With UDI)	Yes	Yes	Yes (11S: No)	Yes (RT ² : No)	No

Figure 3: Comparison of microcomputer operating systems

tasks, and multiple processors — the ability of the operating system to schedule activities becomes a primary consideration. In early multiprogramming operating systems, many of these scheduling routines were time driven. That is, one program would be allowed to execute for a certain time. It would then be interrupted, and another program would be allowed to execute. This time driven scheduling, in effect, forced multiprogramming to occur, but also often involved a significant amount of operating system overhead to manage the process.

Subsequently, the approach of event driven scheduling was developed. With this approach, programs and tasks are allowed to proceed until some predetermined event causes the operating system to interrupt the running task and substitute another. In many applications, event driven scheduling is the most efficient manner of allocating resources. In addition, event driven operating systems can often be modified to include some time driven scheduling routines, where the reverse is not possible. As a result, event driven systems are more flexible and can manage the μ C and other system resources more efficiently.

future issues

As the μ C world continues to evolve rapidly, changes in semiconductor technology will continue to force evolution and improvement in operating systems. As this evolution occurs, a number of trends and developments will influence the user's choice of appropriate operating systems. Some of these developments include:

Very Large Scale Integration (VLSI) Trends. As more complex functions are integrated into μ C chips, operating systems will be required to support a broader variety of needs and application programs. The benefits of VLSI — such as increasing density, substantial improvements in function, and rapidly declining costs per function — accrue to those who rapidly use the newest technologies. As a result, μ C users on the leading edge often can build significant competitive advantages by being first to market.

With regard to operating systems, trends toward greater VLSI integration imply that operating systems should be evaluated based on their ability to most rapidly use technological advances. This ability to capitalize on VLSI trends includes:

- Operating systems with the potential to be integrated into silicon. Intel, for example, has introduced a device that integrates timers, an interrupt controller, and multiprogramming and multitasking operating system "primitives" into one device. (These operating system functions are equivalent to those of the iRMX 86 kernel.) In essence, some of the traditional software functions will become part of the hardware. Such a development opens a number of vistas for future integration.
- Operating systems organized to take advantage of new trends in μ C architecture. One of the most promising developments in this area is object-oriented architecture such as that implemented on iAPX 86 processors under the iRMX 86 operating system and on Intel's new iAPX-432 32 bit micromainframe product family. Essentially, an object-oriented architecture treats different kinds of data and instructions as "objects" and provides common functions that can manipulate objects in a consistent manner. Moreover, the object oriented architecture also meshes closely with the new types of high level languages — such as Ada — now being brought to market. These developments begin to provide μ Cs designed to optimize both program development and execution.

The benefits of VLSI — such as increasing density, substantial improvements in function, and rapidly declining costs per function — accrue to those who rapidly use the newest technologies.

Standards for μ C Languages. μ C applications have evolved requiring higher level languages so less technical users can easily participate in VLSI technology. Without substantial advanced planning, the use of higher level languages can cause significant problems with operating systems. One example of how advanced planning is done is Intel's approach. Intel's iRMX 86 operating system contains both a Universal Development Interface (UDI) and a Universal Runtime Interface (URI). These two interfaces provide a standard upon which high level languages can be both developed and run. In essence, the UDI/URI approach is developing a "software bus" or series of standards for easy and consistent language development. Without such an approach to standardization, each new language could be retarded in its development and each new processor could require a completely new set of compilers.

Among the many benefits of this approach are those of particular interest to persons who have been hesitant to use μ Cs because of the lack of standardized, high-level computer languages. With the development of the UDI/URI approach in iRMX 86, for example, languages currently or soon to be available include FORTRAN 86, PASCAL 86, PLM/86 and ASM 86.

Another major advantage of software standards now being developed is that many vendors can now create languages that will operate effectively on the same operating system. The software bus concept provides these vendors assurance their languages will operate, and it also provides μ C users with a significantly broader range of application languages and even pre-developed software. Among the newer computer languages currently under development by independent software vendors as a result of the UDI/URI standardization are BASIC, COBOL, and "C".

This one development — the standardization of software development and runtime environments — has the potential to be as significant to μ C users as earlier efforts to standardize on communications methods (such as the IEEE 488 standard) or μ C standardization such as the Multibus (IEEE P796). Once a standard is developed and accepted, many different manufacturers can develop products with the assurance that they will be compatible with other products. Hence, the user obtains a broader selection and applications innovation is enhanced. Moreover, users need concentrate only on learning one approach, with a corresponding improvement in speed and productivity of applications efforts.

Standards for μ C Networks. Increasing use of VLSI means that μ C costs per function are declining. As a result, many new applications will become cost-effective. One of the major new application areas VLSI is stimulating is local area networking. The costs of μ Cs and supporting memory are now declining to the point where local and even global area networks make more and more sense. As networking becomes more practical, however, new approaches to networking standards will be required.

Standards are particularly important for networking μ C operating systems, since the operating system will be required to manage many of the networking resources. In this regard, Intel Corp, Xerox Corp, and Digital Equipment Corp have joined together to develop Ethernet, a local area network standard. (Many other firms are also becoming involved with Ethernet, including Hewlett-Packard, Siemens, Nixdorf, Olivetti, and Zilog.)

One of Intel's Ethernet responsibilities is to provide standard modules and standard interfaces for Ethernet users. These include high level, data link layer interfaces to an Ethernet controller on the standard Multibus, a new Multibus Interprocessor Protocol (MIP), and a method to support Ethernet with the iRMX 86 operating system in conjunction with Intel's new Multibus Message Exchange (iMMX). Intel intends to provide VLSI implementations of these standards, up to the data link layer. As further Ethernet standards evolve, the major benefit to networking μ C users will be the ability to take advantage of a wide variety of products with the assurances they will work together effectively.

Protection of Software Investment. Many μ C users have substantial investments in μ C software and are consequently concerned that these investments do not become obsolete before having provided an adequate return on the resources invested. Many current minicomputer users, for example, would like to take advantage of the density, size, and low cost per function benefits of μ Cs but are hesitant to redevelop their existing application software.

Recent developments in μ C software are directed toward solving this concern. The trends toward software and networking standards, for example, will provide a basis for rapid development of new language compilers, and emulators that allow minicomputer users to migrate to more tech-

nically advanced μ Cs without a commensurate reinvestment in software development.

Other trends in μ C operating systems, such as increasing modularity and configurability, also support this direction. Intel's iRMX 86 operating system, following this trend, is developed in modules. This allows future integration of certain modules into silicon as conditions warrant. Modularity and configurability also mean users can eliminate portions of the operating system not appropriate to their application without a performance penalty. In addition, Intel's approach to operating system design means that current μ C systems users can easily and cost-effectively move most of their existing software from 8 bit to 16 bit μ Cs as their application requirements expand. This design consideration, most noticeable in the iRMX 86 operating system, guarantees substantial user software investments are protected while users can, at the same time, rapidly take advantage of the latest developments in VLSI technology.

hardware advances mean change

Rapidly advancing developments in μ C technology are causing a corresponding change in μ C operating systems. More and different operating systems are becoming available as users' needs evolve and become more specialized.

The rapid march of VLSI technology also pressures manufacturers, OEMs, and end users to develop and evaluate operating systems based on their ability to directly translate VLSI advances into applications.

Moreover, the need for a series of operating systems standards is clear. Without standards such as the UDI and URI, operating system development could retard the widespread and fast growing use of productive microelectronics technology. D

November 1981

The iRMX™ 86 Operating System

Bruce Schafer
and Jack Davis
Intel Corporation

Intel's advanced iRMX 86 will not only support the development of software development.

Object-oriented programming languages, such as Pascal, Modula-2, and C, are also supported by this system. Intel's iRMX 86 operating system, following this trend, is designed to be modular. The object-oriented nature of the system allows modules to be developed in isolation. Modules can be developed into either a condition or a function. Modularity and configurability also mean users can eliminate bottlenecks in the operating system and adapt it to their application without a performance penalty. In addition, Intel's approach to operating system design means that current iRMX systems can easily and cost-effectively have most of their existing software from 8 bit to 16 bit iRMX as their application environment. This is a significant advantage.

Noticeable in the iRMX 86 operating system, however, is the substantial cost savings. Investments are protected while users can, at the same time, rapidly take advantage of the latest developments in VLSI technology.

Hardware advances mean change

Rapidly advancing developments in iRMX technology are causing a corresponding change in the operating system. Moving a corresponding change in the operating system is becoming available as users' needs evolve and become more specialized.

The rapid march of VLSI technology and previous manufacturing, OEMs, and end users to develop and evaluate operating systems based on their ability to directly translate VLSI advances into applications.

Moreover, the need for a series of operating system standards is clear. Without standards such as the UDI and ULI, operating system development could trend the wild west and the growing use of productive microcomputer technology.

Standards are particularly important for networking iRMX operating systems since the operating system will be required to manage many of the networking resources. In this regard, Intel Corp., Xerox Corp., and Digital Equipment Corp. have joined together to develop Ethernet, a local area network standard. (Many other firms are also becoming involved with Ethernet, including Hewlett-Packard, Siemens, Microsoft, Olivetti, and Xerox.)

One of Intel's Ethernet responsibilities is to provide standard modules and standard interfaces for Ethernet users. These include high-level, data-link layer interfaces to an Ethernet controller on the standard Multibus, a new Multibus protocol (MIB), and a method to support existing software from 8 bit to 16 bit iRMX as their application environment.

With Intel's new Multibus Message Exchange (MME) Intel intends to provide VLSI implementations of these standards up to the data-link layer. As future Ethernet standards evolve, the major benefit to networking iRMX users will be the ability to take advantage of a wide variety of products with the same iRMX they will work together effectively.

Investment of software investment

Many iRMX users have substantial investments in iRMX software and are consequently concerned that these investments do not become obsolete before having provided an adequate return on the resources invested. Many current microcomputer users, for example, would like to take advantage of the density, size, and low cost per function benefits of iRMX but are hesitant to develop their existing application software.

Recent developments in iRMX software are directed toward solving this concern. The trends toward software and networking standards, for example, will provide a basis for rapid development of new languages, compilers, and emulators that allow users to migrate to more tech-

The overall structure of the iRMX 86 Operating System
... an O.S. designed to exploit the advantages of
Intel's VLSI technology.

INTRODUCTION

Over the past several years, microcomputers have become faster and less expensive. Accompanying this increase in raw horsepower have been enhancements in hardware architecture including high-speed floating-point co-processors, multiple processor support, and soon, Local Area Network controller chips.

This rapid increase in VLSI capability poses a question to microcomputer users: How do they keep up? Already, the cost of developing and maintaining software packages is many times more than the cost of developing the underlying hardware. Fortunately, microcomputer vendors have recognized the problem. Some of the things they can do to help include:

- Provide a wide variety of operating system features as a set of user-selectable building blocks.
- Provide software support for co-processors.
- Provide software that allows the application to take advantage of multiple processors for increased application throughput.
- Provide software that supports the use of the Ethernet* protocol.
- Integrate operating system and hardware functions on the same VLSI component.
- Support standard interfaces that make it easy to move application packages from one operating system to another and to take advantage of future generations of VLSI.

The iRMX 86 Operating System and supporting Intel subsystems provide these capabilities. By taking advantage of them, users can turn the "future shock" of VLSI to their advantage. As such, the iRMX 86 product can be described as a VLSI operating system.

This paper provides an overview of the features of the iRMX 86 Operating System. Associated papers describe how its capabilities are used to support multiple processors [10] and Ethernet [11, 12]. Another paper describes how many of the basic features of the operating system are being integrated with hardware functions [13].

THE iRMX 86 OPERATING SYSTEM

Bruce Schafer and Jack Davis

PURPOSES OF MICROCOMPUTER OPERATING SYSTEMS

Reduced Application Investment

Because software costs are rising while hardware costs are falling, microcomputer customers have discovered that they must carefully account for their software development investment. This accounting must include both the original development and the maintenance of the software. Many microcomputer users have found that it is much cheaper to purchase software rather than develop software from scratch. An operating system allows customers to save a significant amount of time and money in developing their particular application program. By reducing their development costs, applications that would otherwise be unprofitable become profitable.

Improved Portability of Applications

Initial costs of development are not the only major concern for microcomputer customers. The cost of developing new products in an existing or new product line is also a key concern. When one microcomputer board is used to create an initial product offering, another member of the same board family is often used to extend the product line. This is true because Intel is constantly adding new features to its family of boards which offer new hardware functions and increased performance. In order to take advantage of new microcomputer components or boards, customers are interested in using as much of their existing software as possible. The use of an operating system will hide many of the details of the underlying hardware and greatly ease moving an application to a new board.

Maximizing Hardware Utilization

Many applications allow monitoring and controlling more than one device. An operating system can make it appear to the application programmer that he has more than one processor at his disposal. This is accomplished by allowing more than one activity to occur asynchronously. The operating system can go further in providing mechanisms for communication and synchronization between programs running on the microcomputer.

Support of Sound Development Methodology

Modular construction is a key way to control the cost of software development and maintenance. Each module ideally "hides" or encapsulates the effect of major decisions

such as how data is represented. Similar to structured programming, the modular design process permits a complex design to be partitioned into a structure of cooperating modules. This both facilitates top-down development and simplifies the maintenance and evolution of large software systems. Even though users of microcomputer systems are usually familiar with these concepts, the lack of effective tools and tight development schedules often force them to take a choice between modular construction and quick implementation. High level programming languages have played a key role in allowing customers to obtain the best of both worlds. An operating system can take this one step further by adding facilities to simplify modular implementation. For instance, it can allow the customer to identify separate asynchronous activities that his applications must accomplish and write a separate program for each of these activities. The operating system can allow these separate programs to be run as separate tasks and communicate with each other as necessary. When a particular part of the application is to be modified, the change can be isolated to one or a very few number of the original application programs. When additional functions need to be added, additional tasks can be added to the system with little or no change to the existing tasks.

COMMON FEATURES OF MICROCOMPUTER OPERATING SYSTEMS

The varied purposes of microcomputer operating systems have led to some relatively standard features. Following are some of the features provided by the iRMX 86 Operating System.

Multitasking and Multiprogramming

Based on the goals summarized above, a key role of a microcomputer operating system is to allow for multiple programs to execute on the same processor. This maximizes the utilization of the hardware and at the same time provides for modular construction of applications. Most importantly, it allows the application programmer to manage the complexity inherent in real-time applications where multiple asynchronous events are occurring.

When several programs must execute concurrently (or at least appear to do so), an operating system can provide multitasking. Each executing program is called a task. When these tasks must execute in separate environments, the operating system can support multiprogramming to provide these environments. The set of tasks executing in a separate environment can then be called a job.

Interrupt Mapping

Since most real-time operating systems are event driven, they must use the interrupt structure of the underlying hardware. The operating system can provide the mechanism that transforms the hardware interrupts into events that will

cause particular tasks to execute and service the interrupt. In this way each task need be only aware of the interrupt that it is managing.

Timer Support

Real-time applications often require the use of the hardware timer provided by the microcomputer. This may be because they must be aware of elapsed time in order to accomplish their purpose. Another reason is that external events may not occur when they should. The software must be aware that they have not occurred and take corrective action. In either case the programs running as separate tasks may each need a separate timer.

Memory Allocation

Many applications cannot predict their actual memory usage ahead of time. This is because they must deal with an unpredictable environment. A key part of this environment is the operator who invokes various functions of the application. The parameters provided by the operator often affect how much memory the program needs. Besides the operator, the external devices connected to the microcomputer generate a variety of events to which the software must respond immediately. These events are not entirely predictable and thus the amount of memory required to respond to and control these events is not predictable. By providing for dynamic memory allocation, an operating system can help an application adapt to its unpredictable environment without statically allocating the worst-case memory requirements to each part of the application.

Device Support

Often a significant portion of the application development time is spent writing complex code that interfaces the application to vendor-supplied devices. An operating system can provide software that does this interfacing. This feature reduces the customer's development cost and elapsed time.

File Support

Operating systems, in general, use random access devices such as disk drives or magnetic tapes to store and retrieve information. The very simplest of these applications might only store one set of data on each device. Far more common than this, however, is the situation where one disk is to store many different kinds of information. Examples of this information may be parametric information that affects the activity of the application, intermediate data required by the application to accomplish its purpose, and data that the application generates which will be used later.

An operating system must manage the secondary storage space represented by the random access device. This management will usually include support for dynamic allocation of random access space, automatic bookkeeping of this space, and naming each portion used by the application. In

this way the application can treat a single random access unit as if it were many separate devices each of which can be randomly accessed.

Loading Support

Many microcomputer applications involve no mass storage devices and thus all of the operating system and application code resides in read-only memory. For systems that include mass storage devices, this allows some of the code to be loaded into read/write memory and provides some significant advantages to the programmer. These include

- (1) Software can be updated and distributed on disk or similar media.

- (2) If not all parts of the applications must execute concurrently, less total memory may be required if only the code required is loaded into memory.

Human Interface

The vast majority of microcomputer applications involve some interface to a human operator. Many of these applications use standard cathode-ray-tube terminals in order to accomplish this interface. An operating system can reduce the cost of using this interface in three ways:

- 1) By providing for common editing functions that allow the human operator to correct his input before it is seen by the application.

- 2) By providing for the automatic invocation of the correct application program based on input by the human operator.

- 3) By providing functions that make it easy for the application program to determine which parameters have been specified by the operator.

This completes a general description of the features that are provided by iRMX 86. A detailed description of the functional capabilities of this operating system follows.

MAJOR FEATURES OF THE iRMX 86 OPERATING SYSTEM

In order to provide operating system support for applications using the 8086 microprocessor, Intel has developed the iRMX 86 Operating System.[1] Because the system is modular, it allows customers to choose only those features of the operating system that are needed by their applications.

Nucleus

The iRMX 86 Nucleus provides for multitasking, interrupt control, timer support, and intertask communication.[2] While many of these concepts are the same as in other microcomputer operating systems the application interface is

quite different. One reason for this is that iRMX 86 interfaces encapsulate the details of the implementation so that it can be more easily changed without affecting the application. This encapsulation is accomplished by implementing each mechanism as a type manager. Each type manager provides a set of objects that are defined by their attributes and the operations that can be performed by them. The basic object types supported are task, segment, mailbox, semaphore, region, job, and extension. One of these objects, the task, also serves as the subject in the sense that tasks are the active elements of the system and perform operations on all objects.

Tasks. All operations are performed by tasks. A task is an executing program and is characterized by a set of processor register values, a priority, a containing job, and a dispatcher state.

Segments. A segment is a contiguous portion of memory described by a base and a length in bytes. The base of a segment can be loaded into a segment register for use as a code segment, stack segment, data segment, or extra segment.

Mailboxes. Mailbox objects are used by a task when it wishes to pass an object to another task in the system. A set of tasks can use this mailbox to implement synchronization, mutual exclusion, and communication.

Semaphores. Semaphores are used in the place of mailboxes where no actual information needs to be communicated between the tasks. Semaphores have the advantage of requiring less execution time overhead and thus may be a sound alternative where performance is critical. They are also useful in solving resource allocation problems, especially when the number of units of the resource is large or if it is desirable to allocate several units at once.

Regions. The region object type is used to implement critical regions via mutual exclusion. Regions have the advantage of preventing the deletion or suspension of a task during a critical operation. They are also used to guarantee that a high priority task will not wait an excessive amount of time for a resource held by a lower priority task that is currently in a region. This is accomplished by raising the effective priority of the task holding the critical region whenever a higher priority task is waiting for it.

Jobs. Job objects represent the environment in which a set of tasks can operate. This environment is limited by the resources given to an application. Several different things can make it desirable to divide an application into multiple environments:

- 1) Control Dynamic Memory Allocation.

When multiple applications are implemented on a single microprocessor the extent to which each application can

account for the other applications' memory needs is limited. By providing separate memory allocation environments for each application, the user can allocate portions of the total memory to each. In this way he can ensure that one application will not allocate memory to the disadvantage of others. This approach also makes it easier to avoid a key hazard of dynamic allocation, the possibility of memory deadlock.

2) Separately Invoke & Abort Individual Applications

While many applications only require a static set of tasks, some applications involve the dynamic invocation of individual subapplications and require the ability to abort these subapplications at any time. By providing separate environments, the iRMX 86 Operating System allows an individual subapplication to be aborted and have its resources returned to the system without affecting other subapplications in the system.

3) Provide Separate Name Spaces

When multiple applications are run on the same microcomputer, these applications are often designed and implemented by separate programming teams. When these teams select names for external devices that are to be used by their applications, they take the risk of choosing names also used by other applications. By providing a separate environment for each executing application, the names used for each application can be kept separate. At execution time the operator can match the individual names used by the application against the resources provided by the operating system. A particular example of this concept occurs when one program is invoked more than one time concurrently. During each invocation the operator can match a set of devices that the application uses against a particular subset of the devices available. In this way the same program can access several sets of devices, at the same time, because from the point of view of the operating system the program is running in separate environments.

Extensions. The final basic object type is the extension object.[9] The extension object is used by operating extensions to create new types. New objects of the new type can be created as a composite of existing objects. Operators are also provided for deleting a composite object and for obtaining the component objects of a composite object.

The concept of restricting access to objects is an integral part of the iRMX 86 model. When a task creates an object all tasks in its job are automatically given access to the object. A task can pass an object to a task in another job. Two mechanisms for communicating access to an object have been built into the iRMX 86 Nucleus: object directories and mailboxes. The advantage of object directories is that they allow a task to obtain access to an object by knowing only

its name. The advantage of mailboxes is that they also can be used for synchronization and communication between tasks.

To increase the ease in which programs can be debugged, each iRMX 86 function returns an exception code. This code indicates whether the requested operation was successfully performed, and if not, what went wrong. These exception codes may be handled either by instructions immediately following the call to the operating system or by a separate error handler. In the latter case, an error condition will automatically cause control to transfer to a user-provided routine or, by default, to one provided by the system.

Terminal Handler

The Terminal Handler provides a real-time, asynchronous interface between a terminal and tasks running under the supervision of the Nucleus.[3] It can be used either with or without the Debugger. The Terminal Handler provides the following features:

- Line editing
- Multicharacter type-ahead
- Control characters for suspending and resuming output at the terminal
- A means of awakening the Debugger.

The Terminal Handler can be accessed either directly under the supervision of the Nucleus or through the Basic I/O System described later.

Debugger

The Debugger is designed specifically for debugging and monitoring systems running under the supervision of the Nucleus.[4] A special Debugger is very helpful in debugging such systems because their real-time and multi-tasking characteristics render useless many ordinary debugging techniques. The iRMX 86 Debugger is sensitive to the data structures used by the Nucleus and can give "snapshots" of tasks at critical moments. It can also be used to alter the contents of memory. If desired, the Debugger can be included in a debugged application system for troubleshooting in the field. If it is included the Debugger requires only the support of the Nucleus and the Terminal Handler.

Basic I/O System

The iRMX 86 Basic I/O System provides facilities for accessing devices and files residing on random access devices.[5] By taking a modular approach, it allows the customer to choose between support for physical devices such as terminals and random access devices, and sophisticated support for files on the random access devices. It accomplished this goal by providing two types of drivers.

The first are device drivers. Device drivers allow the customer to choose from the set of Intel-provided device drivers in order to support the controllers that are being used. In addition, the customer can add his own device drivers to match custom device controllers.

The second type of driver is called a file driver. The file driver accomplishes goals similar to a device driver in that it is a modular piece of the I/O System which allows the customer just those facilities needed by his application. File drivers implement different types of files.

Named files. The most general type of file provided by the iRMX 86 Basic I/O System is that of a named file. A named file is a byte-oriented random-access file which is given a name to identify it among those on a particular volume. Files can serve as both data files and directories. Directories can point to both data files and directories. In this way a file can be designated by a sequence of file names from the root or main directory on a volume through a sequence of directory files to the actual file being designated. Once located, a file can be opened and closed as many times as desired without further directory searches. This type of file naming mechanism is called a hierarchical file structure.

The accessing of the individual data blocks of a file is designed to both minimize allocation of space on a volume and to minimize the number of disk accesses required to access an arbitrary location in a file. For small files an arbitrary data block can be read with a single physical seek-and-read operation. In large files this can be accomplished with at most, two seek-read combinations. Because there is inevitably a trade-off between space and performance, the Basic I/O System allows the application to specify to what extent space should be compromised for performance or vice versa.

Physical files. The second major type of file provided by the Basic I/O System is the physical file type. Physical files are accessed similarly to physical devices. In order to provide a common interface to a wide variety of devices the interfaces to physical files assumes that any byte on a device can be accessed. The device driver for a particular device then chooses to what extent it supports this file. For instance, a device driver for a line printer would return an error upon a request for seek or read.

Stream files. A third type of file provided by the Basic I/O System is called stream files. A stream file is a sequence of bytes. A task can add bytes to the end of this sequence of bytes and/or read bytes from the front of the sequence of bytes. Any bytes read are consumed by the read operation and thus can only be read once. A key use of stream files is to allow a program that normally writes to a physical device or to a disk file to direct its output to another program.

Because the Basic I/O System provides three basic file drivers which are accessed via a common interface, application programmers do not have to be concerned whether the data that is being output is being written to a physical device, a data file, or directly to another program.

Extended I/O System

The iRMX 86 Extended I/O System[6] builds upon the facilities provided by the Basic I/O System and provides the additional facilities to accomplish two general goals:

- 1) decreasing the cost of implementing applications that use the facilities of the Basic I/O System.
- 2) providing additional features not found in the Basic I/O System.

The fact that the Basic I/O System is designed to be very general means that some of its system calls are overly complex when used in simple situations. The Extended I/O System provides an optional interface that allows calling sequences to be greatly simplified when some of the more sophisticated features of the Basic I/O System are not needed.

One of the additional facilities provided by the Extended I/O System is the notion of logical names. Logical names allow applications to refer to devices without using the actual physical names of the devices. This allows an application to be written for a standard set of devices and then be executed with a variety of different devices. This accounts for both the fact that the user of an application program will vary over time and the fact that the set of available devices will change over time. The Extended I/O System also extends the concept of logical names to include directory files and data files. In this way an application program can refer to a device location without knowing whether it is using an entire physical device, a directory on that device, or a particular data file on that device. Consequently a data file can be substituted for a device like a line printer or a directory on a large disk can be substituted for a smaller disk.

The Extended I/O System provides support for automatic buffering as an optional facility. This support accomplishes two goals.

- 1) Allow the physical accesses to the devices to match its physical characteristics. In particular, it allows the number of bytes requested of the device to match the characteristics of the device such as its sector size.
- 2) Allow the physical access of the device to be concurrent with the execution of the application program. In this way the throughput of the system can be increased by overlapping CPU and I/O time.

The first goal is accomplished by having the Extended I/O System allocate a buffer whose size matches the physical characteristics of the device. Input and output requests are accomplished using the buffer or buffers as intermediate storage. In this way the actual request made to the device controller matches the controller, and is independent of what the application has requested.

The second goal is accomplished by providing one or two buffers where data can be moved to and from at the same time the application is executing. To use the example of sequential input, when the file is open, the Extended I/O System can initiate reads into the buffers that it has allocated for the application. When the application makes its first request, the data it needs may already be in one of those buffers and can be returned immediately to the application. By the time the data in the first buffer is exhausted the second buffer may have been filled. While the second buffer is being used the Extended I/O System can refill the first. In this way, assuming that the execution time required to process a buffer full of information is comparable to the time required to read from the disk, the application will run concurrently with the physical reading of the device. The same approach can be used for sequential output, in this case the physical writing is delayed until the buffer is filled.

By combining the notions of automatic buffer size and automatic overlap, the total throughput of the system can be increased and the execution time of a particular application can be decreased. The Extended I/O System provides both these facilities in a way that makes them appear as if the application is doing a simple sequence of reads and writes. It completely hides the buffering and the overlap algorithm from the application.

Application Loader

The Application Loader uses the I/O System that to load object files into memory[7]. With the loader, you can store some of your code on disk and load it into memory only when you actually need it. This can lower the memory requirements for your application system.

The Application Loader accepts the following types of files:

- 1) **Absolute Files:** The loader places absolute code into memory at predetermined locations.
- 2) **Load Time Locatable (LTL):** These files contain code which the loader can assign to any available memory in the job's memory pool. This version of the loader will automatically update instructions as they are loaded to account for the fact that in a multisection program the code in one segment may refer to code and data in other segments. Since the loader is responsible for allocating

the segments, it can update the code with the appropriate base values while it is loading the code.

The Application Loader also supports overlays. This allows an application to be constructed as a "root" and a set of overlays. This significantly reduces the amount of memory required to support large applications.

Human Interface

The iRMX 86 Operating System provides an additional layer of software called the Human Interface.[8] This layer makes it particularly easy for customers to add customer facilities to the system. It is designed to provide support for interactive commands whose code is usually not resident in memory. In addition to this goal it provides a standard set of commands for the manipulation of files.

In order to make it easy to add custom commands to the system, the Human Interface provides for automatically loading and invoking the appropriate program based on the commands entered by the operator. Once a program is loaded and invoked in this way, it can access its parameters by making a series of calls to the Human Interface routines. These routines will return connections to files as well as other parameters.

Rather than require each customer to implement his own set of basic commands, the Human Interface provides a standard set of file manipulation commands. This set of commands includes commands for renaming files, copying files, displaying a list of the files in a particular directory, creating files, changing access to files, and deleting files.

Bootstrap Loader

The iRMX 86 Bootstrap Loader[7] is used to load the iRMX 86 system and/or application programs into memory from mass storage and begin their execution. It consists of two stages.

The first stage provides a rudimentary device driver and uses it to read in the first part of the second stage. In addition, the first stage may provide a file name to the second stage to identify which version of the system is to be loaded. The first stage may also provide for automatic or manual bootstrap device selection.

The second stage reads the rest of itself in and then finds and loads the operating system. Using the device driver from the first stage, the second stage interprets the file structure on the disk. Since the first stage and device driver handle all the device dependent matters, the same second stage can be used on all iRMX 86 disks regardless of the type of device used. Separately located modules may be combined in a

library, so that the various layers of the operating system and the application may be loaded from one file.

SUMMARY

This paper has outlined the advantages of vendor-supplied operating systems software for microcomputers. Although these advantages parallel those used for operating systems in minicomputers and mainframe computers, the specific facilities required by a microcomputer customer are often different. For example, many microcomputer applications do not require operating system support for devices and files. The iRMX 86 Operating System answers this need by providing layered software. It is organized around a basic Nucleus that supports multitasking and offers I/O facilities as optional layers above this Nucleus. A customer can choose how many of these facilities he requires and then add his application software on top of the operating system.

By taking this approach iRMX 86 reduces the investment required by the customer, improves the portability of applications from one microcomputer to another, allows one microcomputer to be used for multiple concurrent applications, and provides a model that makes it much easier to change and add features.

CONCLUSION

The dramatic increase of computing power provided by microcomputers represents a challenge. How can this power be harnessed without turning the entire labor force into computer programmers? Part of the answer is provided by vendor-supplied operating systems.

By taking advantage of the availability of both high-performance microcomputer hardware and off-the-shelf software, designers can leverage their expertise and investment. End-users can quickly and effectively put microcomputers to use in their applications. Original Equipment Manufacturers can take advantage of what they know best, their market and their technology. In this way, they can play a *leadership* role in taking the microcomputer revolution to their marketplace.

REFERENCES

1. *Introduction to the iRMX 86 Operating System*, Intel Corporation, 1980.
2. *iRMX 86 Nucleus Reference Manual*, Intel Corporation, 1981.
3. *iRMX 86 Terminal Handler Reference Manual*, Intel Corporation, 1981.
4. *iRMX 86 Debugger Reference Manual*, Intel Corporation, 1981.
5. *iRMX 86 Basic I/O System Reference Manual*, Intel Corporation, 1981.
6. *iRMX 86 Extended I/O System Reference Manual*, Intel Corporation, 1981.
7. *iRMX 86 Loader Reference Manual*, Intel Corporation, 1981.
8. *iRMX 86 Human Interface Reference Manual*, Intel Corporation, 1981.
9. *iRMX 86 System Programmer's Reference Manual*, Intel Corporation, 1981.
10. Ronald M. Smith, Multiple Processor Support with the iRMX 86 Operating System, *Proceedings of the iRMX 86 Technical Symposium*, Intel Corporation, 1981.
11. Jack Inman, Getting onto Ethernet with the iRMX 86 Operating System, *Proceedings of the iRMX 86 Technical Symposium*, Intel Corporation, 1981.
12. Ted Forgeron, Intel OEM Building Blocks Support Natural Addition of Ethernet Capabilities, *Proceedings of the iRMX 86 Technical Symposium*, Intel Corporation, 1981.
13. Phillip L. Barrett, VLSI Operating Systems, *Proceedings of the iRMX 86 Technical Symposium*, Intel Corporation, 1981.

REFERENCES

1. Introduction to the iRMK 86 Operating System, Intel Corporation, 1980.
2. iRMK 86 Technical Handbook, Intel Corporation, 1981.
3. iRMK 86 Technical Handbook, Intel Corporation, 1981.
4. iRMK 86 Debugger Reference Manual, Intel Corporation, 1981.
5. iRMK 86 Basic I/O System Reference Manual, Intel Corporation, 1981.
6. iRMK 86 Extended I/O System Reference Manual, Intel Corporation, 1981.
7. iRMK 86 Loader Reference Manual, Intel Corporation, 1981.
8. iRMK 86 Human Interface Reference Manual, Intel Corporation, 1981.
9. iRMK 86 System Programmer's Reference Manual, Intel Corporation, 1981.
10. Ronald M. Smith, Multiple Processor Support with the iRMK 86 Operating System, Proceedings of the iRMK 86 Technical Symposium, Intel Corporation, 1981.
11. Jack Jansen, Getting onto Ethernet with the iRMK 86 Operating System, Proceedings of the iRMK 86 Technical Symposium, Intel Corporation, 1981.
12. Ted Forester, Intel OSM Building Block Support: Natural Addition of Ethernet Capabilities, Proceedings of the iRMK 86 Technical Symposium, Intel Corporation, 1981.
13. Phillip J. Bartlett, VLSI Operating Systems: Proceedings of the iRMK 86 Technical Symposium, Intel Corporation, 1981.

library so that the various layers of the operating system and the application may be loaded from one file.

SUMMARY

This paper has outlined the advantages of vendor-supplied operating system software for microcomputers. Although these advantages parallel those used for operating systems in minicomputer and mainframe computers, the specific facilities required by a microcomputer customer are often different. For example, many microcomputer applications do not require operating system support for devices and files. The iRMK 86 Operating System answers this need by providing a basic operating system that is organized around a basic

July 1982

optional layer above the hardware. A customer can choose how many of these facilities he requires and then add application software on top of the operating system.

By taking this approach, iRMK 86 reduces the investment required by the customer, improves the portability of applications from one microcomputer to another, allows one microcomputer to be used for multiple concurrent applications, and provides a model that makes it much easier to change and add features.

CONCLUSION

The dramatic increase in computing power provided by microcomputers represents a challenge to the power-hungry applications that have been developed for minicomputer programs. The power-hungry applications can be ported to microcomputers by using the iRMK 86 Operating System. The iRMK 86 Operating System can take advantage of the microcomputer's architecture and its technology in the hardware and software layers to provide a model that makes it much easier to change and add features.

An Object Oriented Operating System for Microcomputers

John P. Collins
Miles M. Lewitt
Computer Design



AN OBJECT ORIENTED OPERATING SYSTEM FOR MICROCOMPUTERS

Organizing an operating system around a set of object managers allows designers to tailor the software to the application, not vice versa

by John P. Collins and
Miles M. Lewitt

Operating systems and the various programs and programming systems that run under their control are all tools for manipulating information. Traditionally, these software tools are conceived and implemented as systems of procedures that manipulate data of various types. This is really an extension of the fundamental nature of most computer hardware—work is performed by causing various instructions to operate upon assorted types of data. It is the programmer's responsibility to ensure that the data can be meaningfully manipulated by the instruction.

As the use of computers becomes more widespread, and the complexity of software increases, there is added motivation to discover ways of making the programming task easier, and programs more reliable and adaptable to differing environments. Because of this, a great deal of interest has developed in the various design techniques of programs and programming systems, with most of the emphasis on the structuring of procedures and the relationships among them.

John P. Collins works primarily on the design of the iRMX 286 operating system at Intel Corp., 19440 SW Shaw Ave, Aloha, OR 97005. Previously, he worked on the design of the CP-6 operating system for Honeywell Information Systems; the development of a secure electronic mail service at Information Sciences Institute; and on the design and development of the CP-5 operating system at Xerox Corp. He has a BS in information and computer science from the University of California, Irvine.

Object orientation is a programming methodology that embodies a different view of computer software. In an object oriented system, things happen as a result of interactions between objects—where an object is composed of one or more types of data. An associated object manager performs all of the operations defined on this data. The object is still manipulated by procedures, but is directly manipulated only by the procedures that are a part of the object manager. Information is manipulated in an object oriented system by issuing requests to object managers to perform the desired operations.

Intel's iRMX 86 operating system is an object oriented operating system designed to run on the Intel iAPX 86 and 88 families of microprocessors. It provides several basic object types that implement the fundamental operations required of an operating system. In addition, it allows for the definition of new object types, enabling it to be custom tailored to suit individual needs.

Background

Structured programming methodologies were developed in an attempt to enable programmers to consistently produce more reliable, maintainable software with lower overall costs for program development and maintenance.

Miles M. Lewitt is a member of the Corporate Strategic Staff at Intel Corp. Prior to that, he worked overseas as a software manager to establish Intel's first software engineering group outside of the U.S. Before coming to Intel, Mr Lewitt worked for Honeywell Corp as a developer and program manager for language translators, operating systems, and networks. Mr Lewitt has an MS in computer science from Arizona State University, and a BS in computer science from the City College of New York School of Engineering.

In order to achieve these goals, structured programming methodologies allow large, complex systems to be better understood, through techniques of top down design, decomposition, and stepwise refinement.

Structured programming techniques primarily address the flow of control within systems and individual programs. The decomposition of programs into modules with well defined interfaces is achieved, typically, by breaking out each major processing step into a module, and rigorously defining the interfaces between them: parameters, data structures, table organizations, and the media used to represent the data; all represent decisions that must be made carefully.

This decomposition of programs into easily understood, manageable modules has helped produce programs that are more easily developed, more bug-free, and, in general, more easily maintained. However, because of emphasis on the use of control flow in determining program modularization, it is quite possible to construct a well structured program that is as difficult to modify as a monolithic, totally unstructured program. This can easily occur when a program change, which involves modification of the representation of a program's data is required, or a change is required in the media or its method of access.

Object orientation provides a methodology that extends the concepts underlying conventional structured programming by also structuring the knowledge of how data are represented. It is an approach to program development in which an attempt is made to hide data representation and design decisions that are likely to be changed. This is achieved by an approach to program modularization that minimizes the number of procedures that are aware of how a particular type of data is structured or represented, and that groups these procedures together with their data as a module.

The primary difference between conventional structured programming and object oriented programming is the number of modules directly manipulating each data structure.

Overall structure of a program is still broken down in terms of the steps it involves, but the resulting modules are designed to use data by making requests of object managers rather than manipulate data structures by directly accessing them. Only the procedures directly associated with an object manipulate the object's data structure. Although the data associated with an object may be used by many modules, there is only one module with the necessary knowledge required to access that data directly. This module, called an object's type manager, contains procedures to perform all the manipulations that are required by other modules in the system. As a result, any changes to the structure or representation of the data associated with an object require changing only the procedures in the object's type manager, rather than all procedures in the program which use the data.

Several programming languages and systems now support the object oriented methodology. SMALLTALK, from the Xerox Palo Alto Research Center, is an object

oriented programming system in which object types are referred to as classes and the procedures associated with an object are termed methods. The Ada programming language supports object oriented programming with its package construct. The Intel iAPX 432 processor incorporates object orientation into its basic design. Intel's iRMX 86 operating system is itself constructed as an object oriented system with the ability to support the definition of new object types. This system provides its users with flexibility in dealing with a broad range of possible applications of today's very large scale integration (VLSI) technology.

Methodology

To better understand how object oriented programming differs from the more conventional methodologies, it is helpful to examine the construction of a sample program.

For a sample program, consider a simple card catalog generating program that an average size library might use. Every month, the hypothetical library acquires a number of new books. As each is received, the appropriate card catalog information is entered into a new books file. Each entry begins with the book's subject category (agriculture, physics, fiction, etc) and card catalog number followed by a comma, the author's name, a second comma, and the book's title. Subsequent items of information follow, in a fixed order, with each item separated from the preceding by a comma. The program is run once a month, with the new books file as input. It produces two print files as output. One is a list of card catalog entries destined for manual insertion into the library's subject catalog; these are sorted by subject, alphabetically. The other is a list of card catalog entries destined for the author/title catalog. Each entry in this list begins with either an author's name (last name first) or a book title, and the entire list is sorted alphabetically on the first field in each entry. Using conventional structured programming techniques, an organization of program modules similar to Fig 1 is a probable result.

Processing begins with the input module reading the new books file entries and storing them in memory in the catalog entry table. This is done to make the frequent use of this information as efficient as possible. Next, the subject sort module is invoked. It scans the catalog entry table and produces a subject index table, which contains the numbers (indices) of all the entries in the catalog entry table, sequenced to reflect their alphabetized order when sorted by subject name. This is done in order to conserve memory, since the individual entries in the catalog entry table may contain over 100 characters. The author and title sort modules are then invoked, in turn, to produce similar index tables for alphabetically accessing the catalog entry table by author and title. Next in sequence is the author/title merge module, which merges the author and title index tables to produce a single alphabetized index table and associates a type (author or title) with each index. The final two modules of the program use the subject and author/title index tables, in conjunction with the catalog entry table, to produce properly formatted print files for the two card catalogs.

This implementation follows guidelines for modularization that are common to all methods of structured

programming. The program is broken down into several modules with well defined interfaces and no side effects. Each module performs a function that can be well understood, easily programmed, and verified as to its correct functioning. However, this program fails miserably because it is not easily maintainable in a changing environment. Assume that the hypothetical library grows substantially, receiving many more new books per month, but, for various reasons, cannot run the catalog updating program more frequently. The new books file grows in size so that it frequently will not fit in available memory.

Ideally, a conceptually simple change to the program is all that should be required to solve this problem. Simply have the input module test the size of the file and, if it will not fit in available memory, create the catalog entry 'table' in a disk file, and have subsequent steps in the program access it there. This is not the best solution, because of the substantial effort involved. In fact, this "simple" change will affect every module of the example program significantly, except perhaps for the driver.

Assume that the example program was implemented following an object oriented methodology. Such a program would have much in common with the conventional implementation. The functional decomposition will be the same and the same algorithms and data structures will be used, but the way in which the program is modularized will be subtly, but significantly, different.

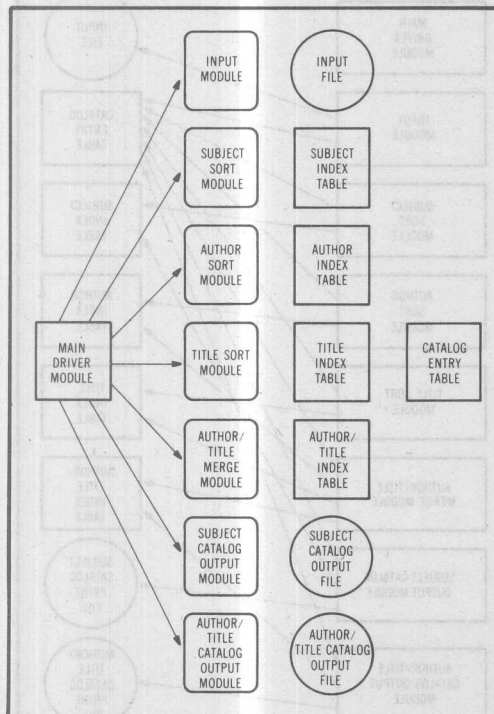


Fig 1 Conventional program structure. Here, main driver invokes use of program modules, and handles program exceptions and storage allocation.

The conventional approach is to split each major processing step into a program module. The object oriented approach augments this process by also using the hiding of design decisions, most notably, the representation of data, as a criterion for modularization.

For example, in the conventional implementation, all of the program modules except the driver directly manipulate the catalog entry table. Thus, all are aware of its structure as an array of formatted lines of text stored in memory. In an object oriented implementation, the catalog entry table would be hidden within a module that also contained all of the procedures required by other modules to use the data represented in the table. For this example, the catalog entry module would support the following operations:

INIT	Defines the catalog entry object as empty, sets number of entries to zero
ADD __ ENTRY (entry)	Appends an entry to the catalog entry object, and increments number of entries
HOWMANY	Returns the number of entries (#ENTRIES) in the catalog entry object
GET __ ENTRY (index)	Returns the indexed entry in the catalog entry object

The subject sort module now implements the subject table object, and supports the following operations:

CREATE __ SUBJECT __ TABLE	Builds the subject table object (formerly the subject index table) by repeatedly calling GET __ ENTRY in the catalog entry module and storing the indices in alphabetically sorted order in the subject table object
GET __ SUBJECT(n)	Returns the text for the nth catalog entry in sorted order by calling GET __ ENTRY with the nth value from the subject table

With a little imagination, the other modules and the operations they must support can be envisioned. For example, the author/title catalog output module would iteratively [from $i = 1$ to $2 * \text{HOWMANY}()$] call the GET __ AUTH __ TITLE function of the author/title merge module to get the appropriate text. It would then format the text according to whether an author or title entry was being made, and finally write it to the author/title catalog print file. Fig 2 illustrates the resulting program structure. The change suggested earlier can now be easily accomplished, with changes to only the catalog entry module.

The primary difference between conventional structured programming and the object oriented programming is the number of modules directly manipulating each data structure. This is illustrated for the two versions of the example program structure methodology in Figs 3 and 4. In Fig 3, information is shared via direct manipulation of shared data structures, conceivably requiring some form of mutual exclusion mechanism. In Fig 4, the object oriented system, information is shared through the common use of services provided by object managers.

Module interfaces in object oriented systems are simpler than those in conventionally structured systems. They consist only of function names with associated parameters. Complex data formats and table structures need never be visible outside of a module. Since object oriented interfaces are easier to specify, and less likely

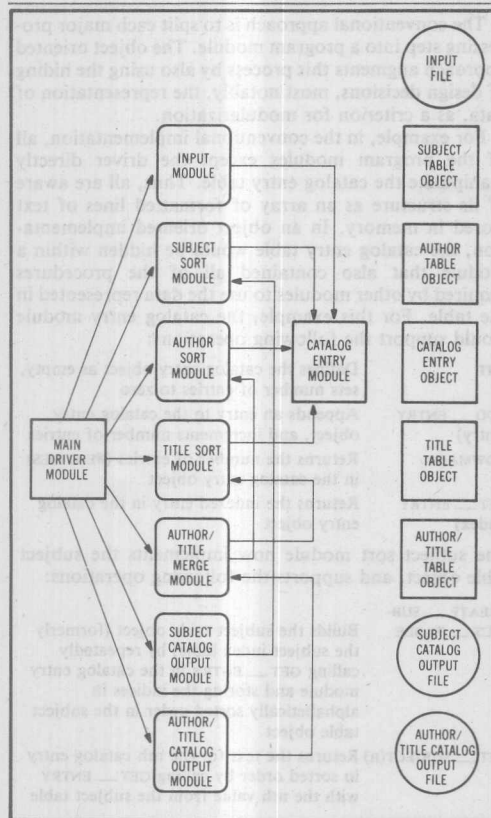


Fig 2 Object oriented program structure. Information is shared through common use of services.

to require change, independent development of the modules of a system may be more reliably and readily undertaken. This capability for easier development of more maintainable software systems is further enhanced since knowledge of global system data structures, or the operation of other modules, is not required in order to understand a given module. Structured programming helps to make programs and programming easier to understand. Object orientation takes another step further in making programming a more manageable undertaking.

Support in the iRMX 86 operating system

The iRMX 86 software package is an operating system for the iAPX 86 and 88 microprocessors. A distinguishing feature of this operating system is the degree of flexibility it provides for configuring and extending instances of itself.

The operating system appears to the user as a collection of supported object types. For each type of object, the operating system provides a set of system calls through which that type of object can be used and manipulated. The object oriented organization provides a framework for learning the system—to understand it, it is only necessary to study the attributes and systems calls for each object type.

Various types of iRMX 86 objects can be divided into two categories: fundamental and composite. All of the fundamental object types are defined within the system nucleus, type managers outside of the nucleus define composite objects. Each type of composite object is defined as a package containing one or more fundamental or composite objects. Implementing a type manager for a composite object requires a way to package the component objects together into a single object, and a way to gain access to the components of a composite object. These functions are provided by the iRMX 86 nucleus.

With a few exceptions, most features of the operating system fit well within the structuring concept of objects. Generally, these exceptions do not involve instances of data structures that could be abstracted into objects. These operating system features are called support functions, an example of which is error handling.

The operating system is structured as five hierarchical subsystems. Innermost is the nucleus, followed by the basic input/output (I/O) system, the application loader, the extended I/O system, and the human interface (see Fig 5). Each of these subsystems contains a collection of highly related type managers and support functions.

Each type manager contains all of the system calls required to manipulate a particular type of object. Always included in the system calls provided by each type manager are those which dynamically create and

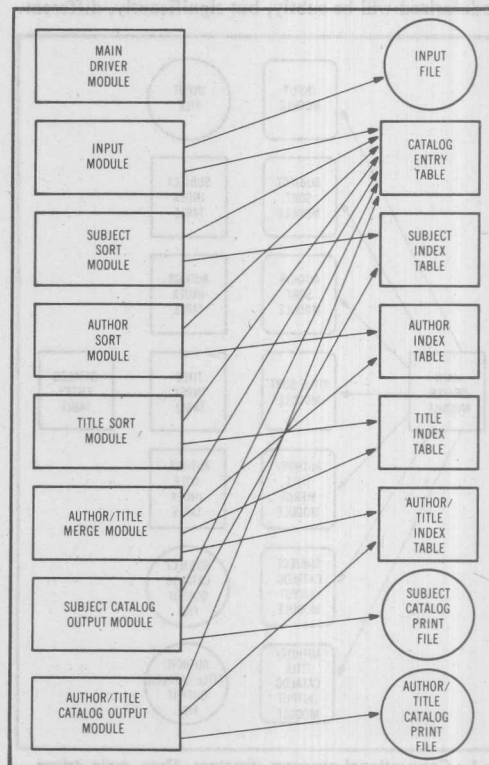


Fig 3 Data manipulation in conventional program structure.

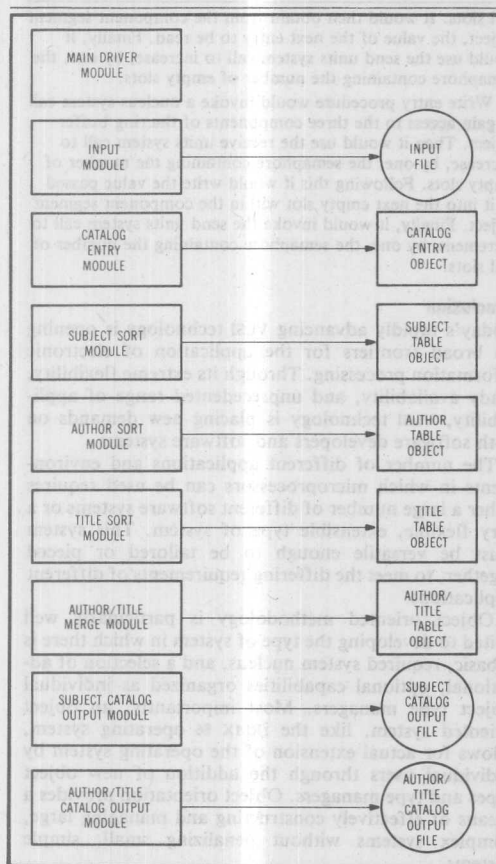


Fig 4 Data manipulation in object oriented structure.

delete an instance of the managed object type. When an object is created, its creator is given a 16-bit integer called a token, which represents addressability to the object. This token is passed as a parameter to any system call that operates on that type of object. The token parameter identifies the particular object to be manipulated.

A special case exists for objects with a type of memory segment. This is the only type of object that has some of its primitive operations implemented in the hardware of the 8086 central processing unit (CPU). A segment is specified to the hardware when its token is loaded into one of the 8086 segmentation registers.

When a system call is invoked to manipulate an object, the operating system can perform various run-time error checks. These checks include determining if the value passed for the token parameter is a valid token, and if the object located by the token is of the type manipulated by the invoked system call. This run-time error checking is implemented as a configurable option.

Object accessibility is an important issue in an object oriented operating system. When an object is initially created by the operating system, only its creator may access it. This creator must take an overt action in order

to share an object. Two means are provided by the operating system to allow an object to be shared. In one mechanism the object is cataloged in a directory within a hierarchical directory structure that the cataloger has access to. Others, who have access to lookup objects in that directory and know the name the object was cataloged under, can gain access to the object. The second mechanism allows one who has access to an object to enqueue it on any accessible mailbox object. Objects are queued on mailboxes in a first in/first out order. When a receive message system call is invoked on a mailbox, the object at the head of that mailbox's object queue is made accessible to the invoker.

This object access model for the iRMX 86 operating system is not enforced on the 8086 CPU because the 8086 lacks access protection features. However, a future implementation of this operating system on an 8086 follow on microprocessor, that supports access protection would enforce its object access mode.

A major goal of the operating system is to provide a very flexible system. One aspect of this is configurability, which is provided by the iRMX 86 structure. Entire subsystems can be included or excluded from a particular configuration. Within a subsystem, individual type managers and support functions can be included or excluded in a particular configuration. At the finest degree of granularity, individual system calls contained in a type manager or in a support function can be included or excluded in a particular configuration. Not all of the possible combinations are permitted or even make sense; however the user of the iRMX 86 operating system can generate a configuration that includes only those features which are needed.

Another aspect of flexibility is to allow the user to easily extend the operating system to contain the additional functionality needed for the specific application. This is accomplished by allowing the user to write subsystems that, just like subsystems provided with the operating system, contain a combination of type managers and support functions. The binding of user written subsystems into the operating system is supported by a set of system calls in the iRMX 86 nucleus. This set of calls are also used by the iRMX 86 supplied subsystems to bind the system together, independent of how the operating system may be configured.

Interfaces between subsystems are purely procedural. Data structures and algorithms used by each subsystem are not visible outside of that subsystem. Thus, once the procedural interface was specified, design and implementation of each iRMX 86 subsystem could proceed in parallel.

This purely procedural interface also provides significant advantages during maintenance and enhancement

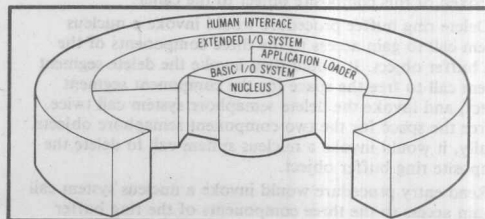


Fig 5 iRMX 86 operating system structure. Each of the five subsystems includes type managers and support functions.

of the iRMX 86 operating system. As long as the procedural interface remains unchanged, the effect of changes to code, algorithms, and data structures is confined. During performance tuning of the system, different algorithms and data structures could easily be tried to determine their effect on performance.

The object oriented approach simplified the developing and maintaining of the operating system. These advantages apply equally well to subsystems written by users of the operating system. Thus, an object oriented approach was chosen for the operating system because of the advantages it provides for developing and maintaining software, and because of the model it provides for building a configurable and extensible system.

Example of an iRMX 86 type manager

Consider implementing a type manager for objects of a type ring buffer. A ring buffer is an N element array into which some tasks write data, and from which other tasks read data. The rate of reading data is independent of the rate of writing data. Data are written into, and read from, the ring buffer in the same sequence. A task desiring to write into a full ring buffer must wait until space is made available by another task reading data from the ring buffer. Likewise a task desiring to read data from an empty ring buffer must wait until another task writes data into the ring buffer.

One implementation of the ring buffer type manager would define a ring buffer as a combination of two semaphore objects and one memory segment object. A semaphore object is a nonnegative integer with two system calls—send units and receive units—through which it can be manipulated. The send units system call increments a semaphore object by a specified value in an indivisible action. The receive units system call will decrement a semaphore object by a specified value in an indivisible action, provided the semaphore remains nonnegative. Otherwise the invoker waits until it is possible. A segment object is an allocated region of contiguous memory. The semaphore and segment objects are two of the fundamental object types implemented by the iRMX 86 nucleus.

The ring buffer type manager would contain four system calls:

- Create ring buffer procedure would invoke the create segment system call to create a segment and invoke the create semaphore system call twice to create two semaphores. It would initialize one of the semaphores to the number of empty slots in the ring buffer (ie, the capacity of this ring buffer object) and initialize the other to the number of full slots in the ring buffer (ie, zero). Then it would invoke a nucleus system call to package the three component objects into one composite ring buffer object, and return the token of this composite object to the caller.
- Delete ring buffer procedure would invoke a nucleus system call to gain access to the three components of the ring buffer object. It would then invoke the delete segment system call to free the space for the component segment object, and invoke the delete semaphore system call twice, to free the space for the two component semaphore objects. Finally, it would invoke a nucleus system call to delete the composite ring buffer object.
- Read entry procedure would invoke a nucleus system call to gain access to the three components of the ring buffer object. Next, it would invoke the receive units system call to decrement, by one, the semaphore containing the number of

full slots. It would then obtain from the component segment object, the value of the next entry to be read. Finally, it would use the send units system call to increase, by one, the semaphore containing the number of empty slots.

- Write entry procedure would invoke a nucleus system call to gain access to the three components of the ring buffer object. Then it would use the receive units system call to decrease, by one, the semaphore containing the number of empty slots. Following this it would write the value passed to it into the next empty slot within the component segment object. Finally, it would invoke the send units system call to increment, by one, the semaphore containing the number of full slots.

Conclusion

Today's rapidly advancing VLSI technology is opening up broad frontiers for the application of electronic information processing. Through its extreme flexibility, ready availability, and unprecedented range of applicability, VLSI technology is placing new demands on both software developers and software systems.

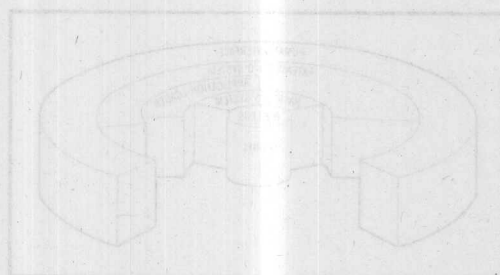
The number of different applications and environments in which microprocessors can be used requires either a large number of different software systems or a very flexible, extensible type of system. This system must be versatile enough to be tailored or pieced together, to meet the differing requirements of different applications.

Object oriented methodology is particularly well suited to developing the type of system in which there is a basic, required system nucleus, and a selection of additional, optional capabilities organized as individual object type managers. Most importantly, an object oriented system, like the iRMX 86 operating system, allows for actual extension of the operating system by individual users through the addition of new object types and type managers. Object orientation provides a means of effectively constructing and managing large, complex systems without penalizing small, simple systems.

Bibliography

- J. Hemenway, "Object Oriented Design Managers Software Complexity," *Electronic Design News*, Aug 19, 1981
- Intel Corp, *Introduction to the iAPX 432 Architecture*, 171821-001
- Intel Corp, *Introduction to the iRMX 86 Operating System*, 9803124-01
- A. J. Maher, "Parnas Partitioning," *Agard Conference Proceedings*, no 251, 1979
- D. L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *CACM*, Dec 1972
- D. Robson, "Object-Oriented Software Systems," *BYTE*, Aug 1981

November, 1982



The IBM AS operating system processor package offers hardware designers a set of thoroughly tested software primitives upon which to build present and future custom hardware designs.

Let Operating Systems Aid in Component Designs

George Heider
OEM Microcomputer Systems
Applications Engineering

LET OPERATING SYSTEMS AID IN COMPONENT DESIGNS

The iRMX 86 operating system processor package offers hardware designers a set of thoroughly tested software primitives upon which to build present and future custom hardware designs

by George Heider

Component users build application systems by integrating standard and custom hardware, software, and packaging. Microprocessors and other very large scale integration components are replacing much custom hardware with larger, more powerful standard hardware modules. Microprocessors lead to powerful systems, but they often require complex system management software. While this complex software often comprises one third or less of the final system software, it may require two thirds or more of the storage development effort. Worse, bugs in system management software sometimes do not show up until late in development or after the product is at the customer's site.

One solution to this problem is to employ standard management software such as operating systems. More complex, multifunction applications in a realtime environment benefit greatly from operating systems. Examples of these applications include file subsystems, public automatic branch exchange (PABX) systems, and transaction processing systems. But implementing

George Heider is a senior applications engineer at Intel's OEM Microcomputer Systems Div, 5200 NE Elam Young Pkwy, Hillsboro, OR 97123. He works primarily with 16-bit software applications, including the iRMX 86 operating system. Previous experience includes telecommunications systems engineering, microprocessor systems, microprocessor operating system development, and disk storage system software. Mr Heider holds an MS in computer science from the University of California, Santa Barbara and a BSEE from Oregon State University.

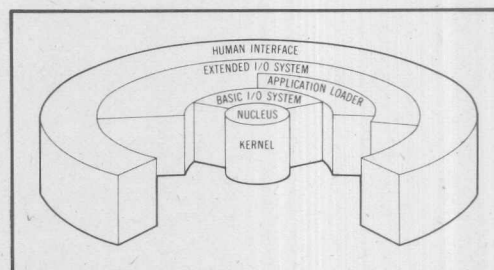


Fig 1 iRMX operating system architecture. Kernel consists of primitives also implemented in hardware in iAPX 86/30 and iAPX 88/30 OSP.

operating system functions in a component design requires new software tools, education, and expertise. Also, these functions are often specific to the particular design, so tools and expertise developed for one application are not suitable for subsequent designs.

These problems are directly addressed by the Intel iAPX 86/30 and iAPX 88/30 operating system processors (OSP) and the iRMX* 86 operating system. The iRMX 86 is a full-featured, realtime multitasking operating system for iAPX 86 or iAPX 88 based systems. The Intel OSP implements the iRMX 86 kernel functions in hardware consisting of an iAPX 86 or iAPX 88 central processor coupled with an operating system firmware (OSF) component, the Intel 80130. The OSF extends the base iAPX 86 and iAPX 88 architecture by adding 35 operating system primitive instructions to the base iAPX 86 or iAPX 88 instruction set; systems can be built directly on the OSP. System implementation time is thus decreased by having fully debugged operating system functions in hardware.

iRMX™ is a trademark of Intel Corp

Further capabilities can be added by extending the set of OSP primitives or by integrating portions of the iRMX 86 on top of the OSP.

Operating system architecture

The iRMX 86 architecture shown in Fig 1 consists of the nucleus and layers for the basic input/output (I/O) system, extended I/O system, application loader, and human interface. The system also provides a debugger, a terminal handler, a bootstrap loader, and a patch facility.

While the nucleus is the lowest layer of the operating system, fundamental system functions are handled by the nucleus kernel, which is the core of any operating system. The kernel controls memory allocation, allocates processor resources, communicates between processes, and manages interrupts. In the Intel OSP these functions are implemented in hardware. (OSP functions are described in Table 1; additional functions supported by the iRMX 86 nucleus are shown in Table 2.) Software development can be based on either the OSP or the iRMX 86, allowing software development to proceed in parallel with hardware development.

In addition to the operating system primitives, the OSP contains timers and interrupt control logic expandable from 8 to 57 interrupt levels. The timers include a system clock, user timer, and baud rate generator. The 40-pin OSP has bus buffers and demultiplex logic, which allows it to interface directly to the iAPX 86 or iAPX 88 multiplexed bus. The OSP can be located at any 16k-byte address boundary in the 1M-byte system address space. Application interface to OSP stepping and revision levels is independent. A block diagram of the 80130 is shown in Fig 2.

Minimum hardware requirements for the iRMX 86 operating system shown in Fig 3 are 1.8k bytes of random access memory (RAM), about 16k bytes of kernel code memory, and integrated circuits. By comparison, the OSP shown in Fig 4 still requires 1.8k bytes of RAM, but does not require the kernel code, the programmable interrupt controller, or the programmable interrupt timer. These are all replaced by the OSP. Approximately 1k bytes of required system configuration code are not shown in Figs 3 and 4.

Kernel functions

Since it defines system architecture, application requests for system operations like interrupt management and memory allocation must go through the kernel. These

TABLE 1
OSP primitives

Primitive	Description
JOB	
CREATE JOB	Creates a job partition including memory pool, task list, and stack area.
TASK	
CREATE TASK	Creates a task with specified environment and priority. Task is created in ready state. Checks for insufficient memory available within containing job.
DELETE TASK	Deletes a task from system as well as from any queues it is awaiting. Task's state and stack segment are deallocated.
SUSPEND TASK	Suspends a task (changes its status to suspended) or increases task's suspension count by 1. A sleeping task may also be suspended and will awaken suspended unless resumed.
RESUME TASK	Decreases suspension count of a task by 1. If at that point count is reduced to 0, task state is made ready. If it was suspend-asleep, it is put back to sleep.
SLEEP	Puts task in asleep state; up to 10 ms units can be specified.
GET TASK TOKENS	Gives token for a task or task's job partition.
INTERRUPT	
SET PRIORITY	Changes task's priority to value passed in primitive.
SET INTERRUPT	Assigns an interrupt handler to a level. Task that makes this call is made interrupt task for same level, unless call indicates there is no interrupt task.
RESET INTERRUPT	Disables an interrupt level; cancels interrupt handler; deletes interrupt task for level if assigned.
GET LEVEL	Returns number of the interrupt level for highest priority interrupt handler currently in operation (several interrupt handlers can be operating).
EXIT INTERRUPT	Completes interrupt processing and sends end of interrupt signal to hardware.
SIGNAL INTERRUPT	Invokes interrupt task assigned to a level from that level's interrupt handler.
WAIT INTERRUPT	Suspends interrupt task state pending a signal interrupt from an interrupt handler. Used by an interrupt task to signal its readiness to service an interrupt.
ENTER INTERRUPT	Sets data segment base for an interrupt handler.
ENABLE	Enables external interrupt level.
DISABLE	Disables an external interrupt level.
GET EXCEPTION HANDLER	Reads location and exception handling mode of current OSP exception handler for a task.
SET EXCEPTION HANDLER	Establishes location and exception handling mode of current OSP exception handler for task.
SIGNAL EXCEPTION	Notifies current OSP exception handler of exception.

requests are made by system calls, or primitives, which are comparable to subroutine calls for system actions. Since the kernel manages much of the system hardware, the application code need not concern itself with many hardware details. This independence is not absolute, however: system hardware or resources not managed by the kernel still require application code.

Basic kernel concepts can be explained using a general purpose system (Fig 5). Input data can be characters, analog signals, or digital signals; processing can be numerical analysis, editing, spectrum analysis, process control algorithms, or virtually any other transformation. Processed data must be sent to an interrupt driven output device—a display, a communications line,

Primitive	Description
SEGMENT	
CREATE SEGMENT	Dynamically allocates area of memory of specified length in 16-byte paragraph units up to 64k-byte maximum (eg, for use as buffer). Returns location token for segment allocated.
DELETE SEGMENT	Deallocates memory segment indicated by parameter token.
ENABLE DELETION	Allows deletion of system data type value indicated by location token.
DISABLE DELETION	Prevents deletion of system data type value indicated by location token.
MAILBOX	
CREATE MAILBOX	Creates a mailbox with specified task queuing discipline. Returns location token.
DELETE MAILBOX	Deletes a mailbox and returns its memory. If tasks are waiting for mailbox, they are awakened (ie, their state is made ready) with appropriate exception condition. If messages are waiting for tasks, they are discarded.
SEND MESSAGE	Sends message segment to mailbox.
RECEIVE MESSAGE	Task is ready to receive message at mailbox. Task is placed on mailbox task queue. Task can wait for response indefinitely, wait (generally 10 ms) units, or not wait. When complete, primitive returns to task the location token of message segment received.
REGION	
CREATE REGION	Creates region data type value, specifying queuing discipline. Returns token for region.
DELETE REGION	Deletes region if the region is not in use.
ACCEPT CONTROL	Gains control of region if region immediately available, but does not wait if not available.
RECEIVE CONTROL	Same primitive as accept control but task that performs it may elect to wait.
SEND CONTROL	Relinquishes region.
OTHER	
SET OS EXTENSION	Links new primitive with kernel.
GET TYPE	Gives system type code of a system data type.

control hardware, or mass storage. In this general purpose system, input, process, and output are the only functions, or tasks, that make up the system.

Buffer management

Assume input data will be placed into 128-byte buffers by the input task. Without help from the operating system, the buffers must be prelocated in RAM. Software is needed to manage the buffers, which must be given to the tasks in the correct sequence and returned for reuse when empty. If the buffers are too small, or if RAM is moved, the software must handle these changes.

If an operating system or OSP is used, the locations and sizes of RAM are made known to the kernel during

system initialization. The input task requests each buffer, or memory segment, from the kernel by making the kernel system call "create segment" with 128 bytes. If a larger buffer is needed, the create segment call needs a larger value for the size parameter. When the buffer is full, the input task gives the segment to the process task. When the buffer is no longer needed, it can be returned to the system memory pool by a "delete segment" system call. Because the kernel dynamically manages memory allocation and buffer access, no additional code for these functions is necessary.

Communication and synchronization through mailboxes

The sample system needs a dispatching algorithm to send the segments from task to task. Such an algorithm can be written without an operating system. For example, the input task can fill a buffer and call the process task. When the process task finishes, it can call the output task; the output task can finish with the buffer and return. When control returns to the input task, system processing for that buffer is complete. Another method is to have a polling task occasionally check if buffers are ready to be sent to other tasks. Both methods are inefficient and rigid, requiring that each task finish processing data in each buffer before another task can run.

With an operating system, the buffers can be sent from task to task through "mailboxes"—places where tasks can send or receive data. (See Fig 6.) Task A sends a message (segment) to mailbox 1 and specifies mailbox 2 as a return mailbox. Task A then waits for a return message at mailbox 2. Task B receives the message (segment) from mailbox 1, then sends a return message with status to mailbox 2.

Task A receives the return message, which contains task B status, and synchronizes the two tasks.

In general, each task obtains a segment, modifies its contents, sends the segment to the next task, and waits for another segment. The input task first gets a segment using "create segment." When the segment is full, the input task uses the kernel call "send message" to send the segment to mailbox A. The process task uses the "receive message" system call to wait at mailbox A for the segment. The process task receives the segment, processes the data, puts the new data in the segment, and sends the segment to mailbox B. The process task then waits at mailbox A for the next segment from the input task. The output task takes the segment from mailbox B

TABLE 2

Additional primitives supported by the iRMX 86 nucleus

CATALOGING SYSTEM**DATA TYPES**

CATALOG OBJECT	Catalogs system data type token under name given by task in job partition directory.
UNCATALOG OBJECT	Removes name and token from job partition directory.
LOOKUP OBJECT	Uses name to find token cataloged in job partition directory.

NEW SYSTEM**DATA TYPES**

CREATE EXTENSION	Notifies kernel of new system data type code for new system data type.
DELETE EXTENSION	Removes system data type code and deletes all composite system data types with that system data type code.
CREATE COMPOSITE	Creates new system data type from list of current system data types and system data type code received from create extension.
DELETE COMPOSITE	Deletes new system data type.
INSPECT COMPOSITE	Gives list of system data types that form new system data type.
ALTER COMPOSITE	Changes list of system data types that form new system data type.

SEMAPHORES

CREATE SEMAPHORE	Creates semaphore system data type.
DELETE SEMAPHORE	Deletes semaphore system data type.
SEND UNITS	Task adds a number of units to semaphore.
RECEIVE UNITS	Task asks for a number of units from semaphore. Task can wait for response indefinitely, wait (generally 10 ms), or not wait.

OTHER**PRIMITIVES**

GET PRIORITY	Gives priority level of task.
FORCE DELETE	Deletes system data type even if disabled delete has been called for system data type.
GET SIZE	Gives byte size of memory segment.

ADDITIONAL JOB**PRIMITIVES**

OFFSPRING	Returns child job partitions created by a task in parent job partition.
GET POOL ATTRIBUTES	Gives memory pool attributes of job partition, including pool minimum, pool maximum, initial size, number of bytes used, and number of bytes available.
SET POOL MINIMUM	Changes pool minimum for job partition.
DELETE JOB	Deletes job partition and returns its memory to parent job partition.

and outputs the data. The output task has two choices: it can either delete the segment, letting the input task create more segments, or it can send the segment to mailbox C. After sending or deleting the segment, the output task waits at mailbox B for the next segment from the process task. If the output task sent the segment to mailbox C, the input task segments from the output task, synchronizing the input task with the output task. If the output task deleted the segment, the input task creates a new segment and waits for input data. The entire process runs continuously, synchronized by mailboxes and segment availability. Additionally, the tasks can elect to wait for a specified amount of time at mailboxes, and if no segments

appear, an error routine can alert the system operator that processing has stopped.

The mailbox method has several advantages over synchronization algorithms and polling tasks. The entire process is synchronized by the availability of data in segments, eliminating the need for algorithms and extra code; the same process applies whether the tasks operate at the same or different speeds. Also, burst input or output rates can be handled by adding buffers. For instance, if too much data arrives for the process or output tasks to handle immediately, the input task fills multiple buffers and passes them to mailbox A. The process task takes each segment in turn. After processing is completed, the segments are all sent to mailbox C, and the process waits for the next burst of data. The only interfaces between the tasks are mailboxes and segments, so tasks can be easily replaced or added to the processing loop; the same scheme works for larger or smaller segments.

Tasks and task scheduling

Tasks are independent bodies of executing code, initialized and scheduled by the kernel. Therefore, tasks must have iRMX 86 parameters like priority, initial memory resources, entry address, and other iRMX 86 data. A task is like an expanded subroutine managed by an operating system. The actual application code is written much the same as it is without an operating system except that requests are made using kernel calls.

Even though the system's multiple independent tasks appear to run simultaneously, only one task actually runs at one time. Some method of scheduling is needed to decide which task receives control of the system processor; this scheduling depends on the task priority. Since data coming into a system must not be missed, the input task has the highest priority. Data going out of the system are next in importance, so the output task has second priority; the sequential process task has the lowest priority. The scheduling algorithm is simple—the highest priority task that is ready to run will get control of the processor. This is an example of preemptive priority. In this case, ready to run means that a task is complete—it has a segment to fill and data coming in (input task), data to process (process task), or data to output (output task). For instance, if input data arrives when the process task is running and the input task has a buffer waiting for data, the input task will preempt the process task to receive

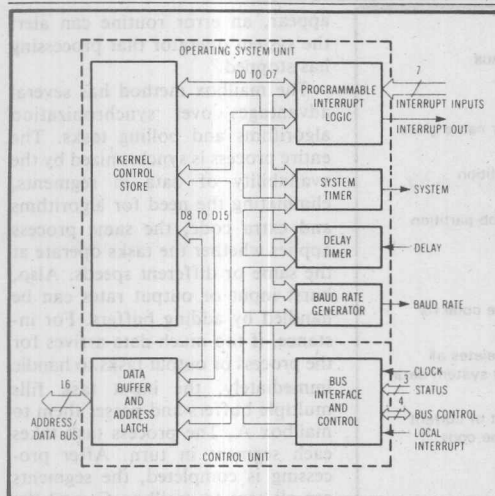


Fig 2 80130 firmware component performs clock and interrupt control functions, and supplies operating system primitives.

the data. If the input task is not running and the hardware driven by the output task is ready to output another data value, the output task will receive control of the processor.

Since the operating system schedules the tasks, each task is designed as though it has sole control of the processor. Tasks make system calls such as receive message, which may cause another task to run because no message is waiting. In addition, interrupts will likely cause a different task to run. The kernel can schedule the tasks because only interrupts or system calls can cause a higher priority task to become ready, and both of these are handled by the kernel. Thus any time an interrupt occurs or a system call is made, the kernel runs the highest priority task that is ready. The tasks are written without any code to manage scheduling. The kernel scheduling is general purpose, so adding new tasks to the system does not require modifying the

scheduling functions. A system with work balanced among the tasks runs as though all tasks perform simultaneously.

The net result of task scheduling is that the system runs as fast as it can. When data come in, the input task will always get control of the processor. The output task will execute whenever it has data to send and the input task is not running. The process task will run whenever it has data and no other tasks are running. Also, tuning the system is easier with the standardized mailbox interfaces: slower tasks can be easily removed and replaced with faster tasks, and remaining tasks will not be affected.

In a multitasking system, multiple independent tasks execute concurrently. Buffer transfers occur through mailboxes rather than through a direct interface to tasks, and system functions not related to the primary data processing functions can be handled by other tasks. For example, a supervisory task that monitors a system console for operator requests can be added to the system at a lower priority than the process task. No changes to any scheduling algorithm would be required.

Interrupt management

The iRMX 86 kernel and the OSP provide two classes of interrupt management: interrupt handlers and interrupt tasks. An interrupt handler is a short procedure whose only function is to respond to the interrupt as quickly as possible. All interrupts become disabled in order to let the interrupt handler execute at top speed. Interrupt handlers can make only a few system calls. In the sample system, the interrupt procedure receives a data value, places it in a buffer, and returns. When the buffer is full, the interrupt handler notifies the interrupt task. Typical response time for an 8-MHz iAPX 86 processor, from the time an interrupt occurs until the interrupt handler gets control is 30 to 50 μ s. In the unlikely event of a worst-case time, response time is about 160 μ s.

Higher priority interrupts are enabled when an interrupt handler gets control, is 30 to 50 μ s. In the unlikely task uses a mailbox to pass the full buffer on to the next task. Since both interrupts and tasks have priorities assigned to them, the kernel uses the task priority to

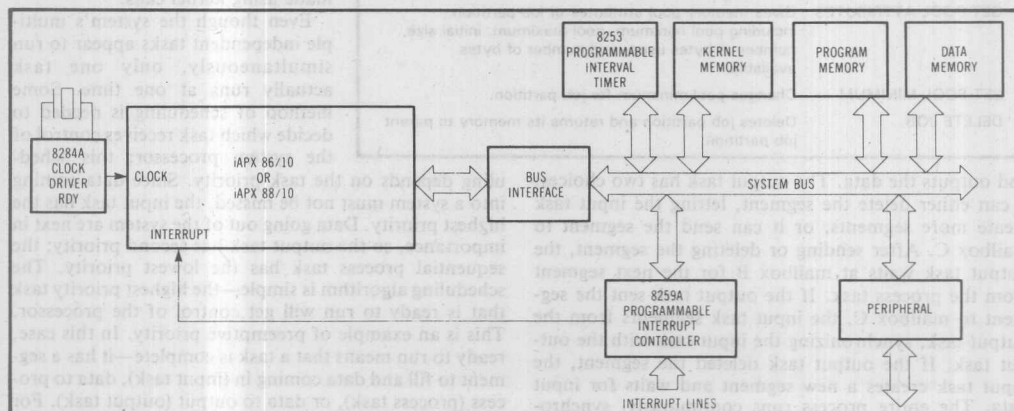


Fig 3 iRMX 86 hardware requirements. Operating system processor fits into basic hardware system for iRMX 86 and brings with it functions of kernel memory, 8259A programmable interrupt controller, and 8253 programmable interrupt timer.

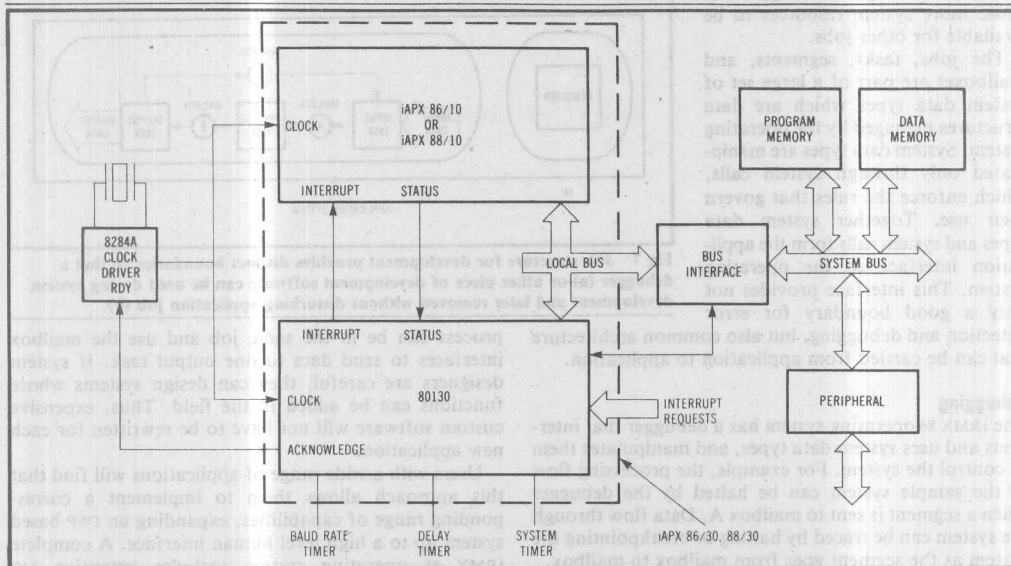


Fig 4 Basic hardware system with iAPX OSP. OSP replaces kernel code, programmable interrupt controller, and programmable interval timer.

determine if interrupts should be disabled or enabled. If the task priority is higher than an interrupt priority, that interrupt is disabled while the task is running. A priority level can be given to a task that disables all, some, or none of the interrupts: ie, defining a task that is more important than all interrupts (initialization task), more important than some interrupts (input task), or less important than all interrupts (processing task).

Multiprogramming

System parameters in a component system are normally well defined: RAM locations are fixed, code addresses are known, and address and I/O ports are specified. Application code usually depends on these parameters. If the system changes, substantial alterations are often needed in the application code. However, if an iRMX 86 operating system is used, the kernel is made aware of system resources during system configuration. System configuration assigns these resources to "jobs."

Jobs do not do work but instead serve as resource boundaries, containing tasks that accomplish system functions. Many component applications systems, including the sample system, will have only one job. All system resources are given to the job and all tasks are contained there. When the system is initialized, the job is created and control is passed to the first task in the job.

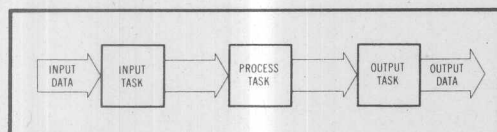


Fig 5 General purpose system consists of 3 basic functions. Application code receives data, places data in buffer, then processes it. Processed data are sent to interrupt driven output devices.

Multiprogramming occurs when a system has two or more jobs. The system boundaries provided by jobs confine errors and define limits for system resources such as memory. These boundaries limit the effect of one job on another. For instance, the system debugger is a separate job. During development, the sample processing system would look like Fig 7. After development, the debugger would be removed, leaving only the application system. The job environment of the processing system is not affected by adding or removing the debugger. The overall system will, of course, be affected

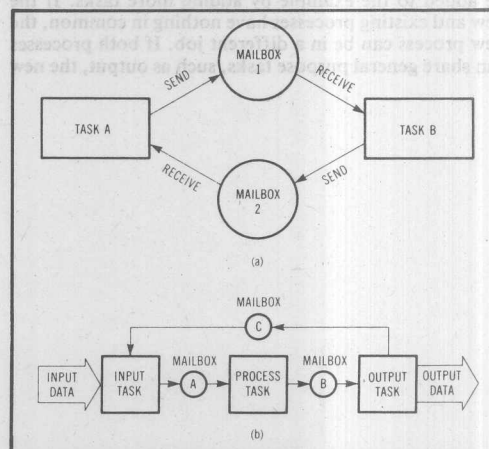
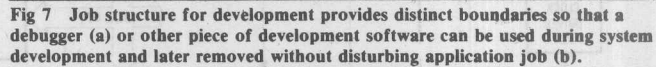


Fig 6 Mailboxes allow intertask communication by providing places to send and receive messages (a). Synchronization is easy since tasks can poll a mailbox and wait for messages. Mailboxes also form interfaces between tasks in application system (b) so tasks can be easily added or removed without changing code.

The jobs, tasks, segments, and mailboxes are part of a large set of system data types which are data structures managed by the operating system. System data types are manipulated only through system calls, which enforce the rules that govern their use. Together system data types and system calls form the application interface to the operating system. This interface provides not only a good boundary for error detection and debugging, but also one that can be carried from application

The iRMX 86 operating system has a debugger that interprets and uses system data types, and manipulates them to control the system. For example, the processing flow in the sample system can be halted by the debugger when a segment is sent to mailbox A. Data flow through the system can be traced by halting or breakpointing the system as the segment goes from mailbox to mailbox.

Conclusion



Users with a wide range of applications will find that this approach allows them to implement a corresponding range of capabilities, expanding an OSP based system up to a high level human interface. A complete iRMX 86 operating system includes extensive I/O capabilities, a debugger, an application loader, a bootstrap loader, and integrated user console functions. Such a system can perform general purpose processing and still provide all iRMX 86 facilities. With these features, one operating system can be used for current projects and expanded for future ones, minimizing software learning curves for new applications.

Introduction to the iRMX 86 Operating System, no 9803124,
Intel Corp, Santa Clara, Calif, 1982

iRMX 86 Nucleus Reference Manual, no 9803122, Intel Corp,
Santa Clara, Calif, 1981

*Using the iRMX 86 Operating System on iAPX 86 Component
Designs*, Application Note AP110, Intel Corp, Santa Clara,
Calif, 1981

J. Zarella, *Operating Systems Concepts and Principles*,
Microcomputer Applications, Suisun City, Calif, 1979

Memory Expansion Boards

6

A PRIMER ON MAGNETIC BUBBLE MEMORY

Magnetic bubble memory is a solid-state technology with high reliability, ruggedness, small size, light weight, and limited power dissipation. It has applications in telecommunications, data acquisition, industrial control, terminals, and small business computers. Yet many potential users remain unsure of the nature of a bubble memory. This primer is intended to introduce these users to the technology.

What a Magnetic Bubble Memory Is

A magnetic bubble memory stores data in the form of cylindrical magnetic domains in a thin film of magnetic material. The presence of a domain (a bubble) is interpreted as a binary 1, and absence of a domain is a 0. Bubbles are created from electrical signals by a bubble generator within the memory, and reconverted to electrical signals by an internal detector. Externally the memory is TTL-compatible.

An external rotating magnetic field propels these cylindrical domain bubbles through the film. Metallic patterns or chevrons deposited on the film steer the domains in the desired directions. Transfer rates, once started, are in the tens of thousands of bits per second, but because the data circulates past a pickup point at which it becomes available to the outside world, there is a latency averaging tens of milliseconds before data transfer can begin. In these respects, magnetic bubble memories are serial high-density storage devices like electromechanical disk memories. However, in a disk, the stored bits are stationary on a moving medium, whereas in the magnetic bubble memory the medium is stationary and the bits move.

Advantages of Magnetic Bubble Memories

The principal advantage of magnetic bubble memories are their non-volatility—that is, if power fails, the stored data is retained. Products incorporating bubble memories therefore do not require battery backups. Magnetic bubble memories share this feature with read only memories (ROMs), erasable PROMs (EPROMs), and electrically erasable PROMs (E²PROMs). However, unlike any of these technologies, magnetic bubble memories can have data written into them at any time, at speeds comparable to those at which data is read. Furthermore, unlike disk memories, bubble memories are quiet and very reliable, because they have no moving parts. They are compact, and they dissipate very little power. Their support circuits are compatible with microprocessor systems. With a million or more bits per device, a bubble memory can store 16 to 64 times the amount of data of alternative semiconductor memories, providing very high storage capability in a compact space. Bubble memory has a wide variety of applications, some of which are listed in Table 1.



Figure 1. Magnetic Domains in Thin Film Under Increasing Magnetic Field

Table 1. Bubble Memory Applications

Numerical control	Robotics
Process control	Oil exploration
Aircraft navigation	Data acquisition
Cable television	Portable instruments
Telecommunications terminals	Avionics
Point-of-sale terminals	Gasoline pumps
Private branch telephone exchanges	Personal computers
Word processors	Office equipment
Flight-line test equipment	Automatic test equipment
Data encryption	

How Bubbles are Formed

Magnetic domains are found in all kinds of magnetic materials—iron bars, the coating on magnetic tape, ferrite toroids (the most common form of computer memory in the 1960s). Each domain is a group of atoms with parallel magnetic orientations. When the material in bulk is unmagnetized, the domains are oriented at random in three dimensions. When the material is magnetically saturated, most of the domains have the same orientation. Magnetization to a level less than saturation orients some of the domains to a common direction, but leaves many of them randomly oriented. When a domain orientation changes, usually by imposing an external magnetic field, the domain itself does not physically move, but boundaries between domains that have different orientations move or disappear altogether.

In an extremely thin film, less than 0.001 inch thick, the domain orientations may be constrained to two dimensions. In some kinds of material (orthoferrites and garnets), with proper crystallographic orientation, the domain orientations are always perpendicular to the film. When these materials are not in a magnetic field, some domains are oriented upward and some downward (north magnetic poles of some domains are on top of the film, and those of other domains are on the bottom). In these materials, the magnetic domains tend to be long and snakelike in the absence of an external field (Figure 1). When a weak magnetic field is applied perpendicular to the film, the domains that are oriented opposite to the applied field become substantially narrower. As the applied field, called a *bias field*, is made stronger, the length continues to decrease, until it becomes approximately the same as the width. Each domain is now cylindrical, magnetized oppositely to the applied field, and immersed in a much larger domain that is magnetized in the same direction as the field.

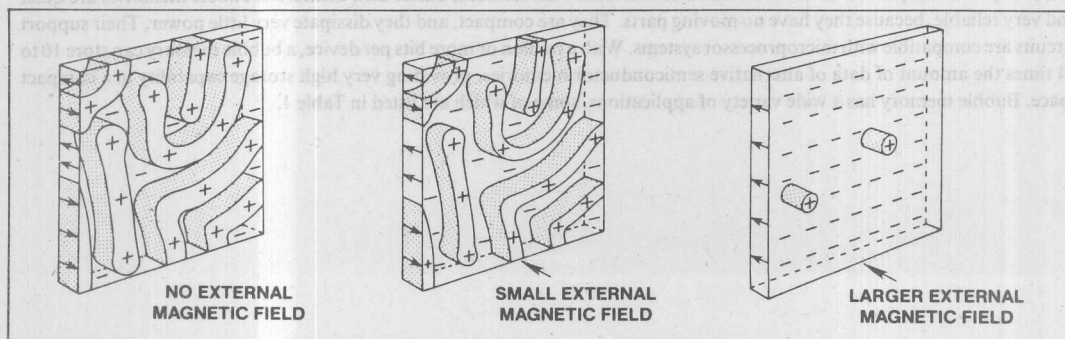


Figure 1. Magnetic Domains in Thin Film Under Increasing Magnetic Bias Field.

These small domains are the bubbles, generally less than 3 micrometers ($1/10,000$ inch) in diameter (Figure 2). When they are viewed from above, only the round shape is apparent, giving the domains the appearance of bubbles. If the bias field were to be made still stronger, all the bubbles would shrink and then disappear altogether; the entire film would be magnetized in the same direction as the bias field. The effect is reversible—that is, if the bias is removed, the domains return to a snakelike form.

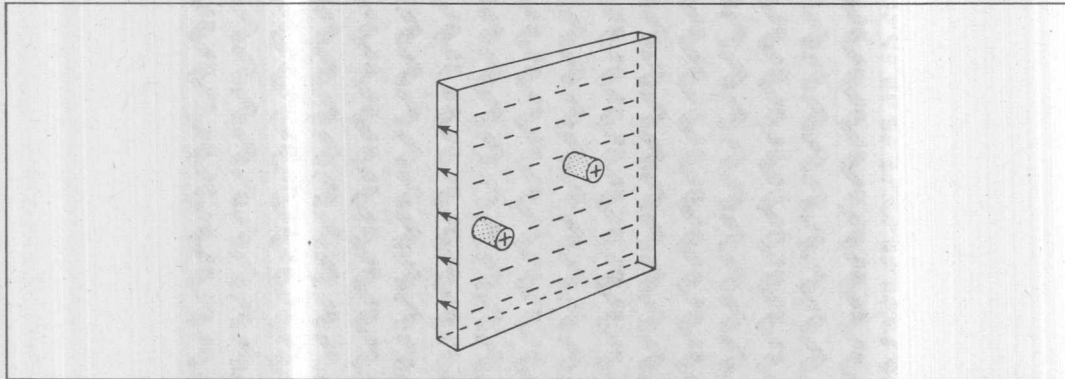


Figure 2. Magnetic Bubbles in a Thin Film

Why a Bubble Moves

Magnetic bubbles will move if they are in a magnetic field gradient. For instance, it will move from a region of lesser magnetic field strength to a region of greater strength. This is similar to the way a nail is pulled to the end of a bar magnet when it gets close the magnet.

In a bubble memory a magnetic film pattern is overlaid on the layer of bubbles. When this layer is magnetized it pulls the bubbles to the points of greatest field strength (or poles) as in Figure 3. The bubbles could then be moved if the pattern elements were moved.

A more easily controlled magnetic field is generated by two coils wrapped around the bubble layer and magnetic film pattern. With appropriate specification of the current in two coils positioned at right angles, the direction of the poles on the stationary elements can be changed in a controlled manner.

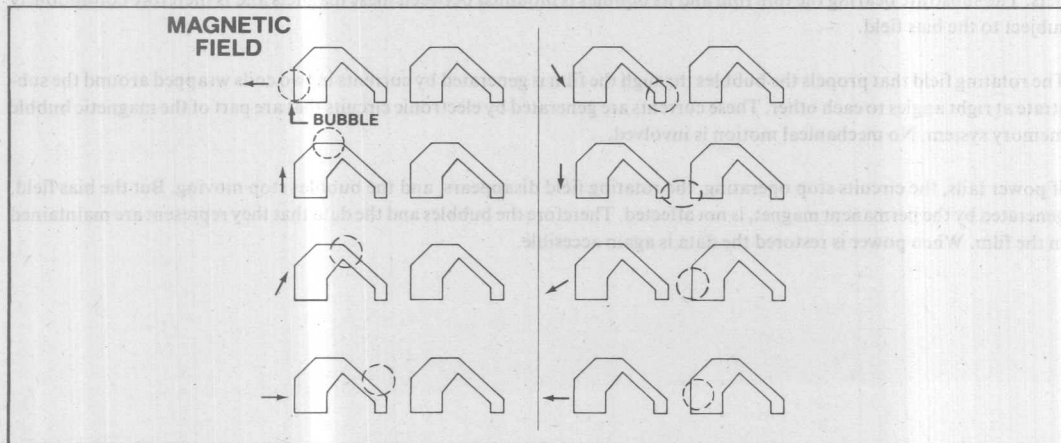


Figure 3. Bubble Propagation Under Asymmetric Chevrons

Various shapes for these metallic patterns have been used by different manufacturers to control the movement of the bubbles. At Intel asymmetric chevrons are used (Figure 3).

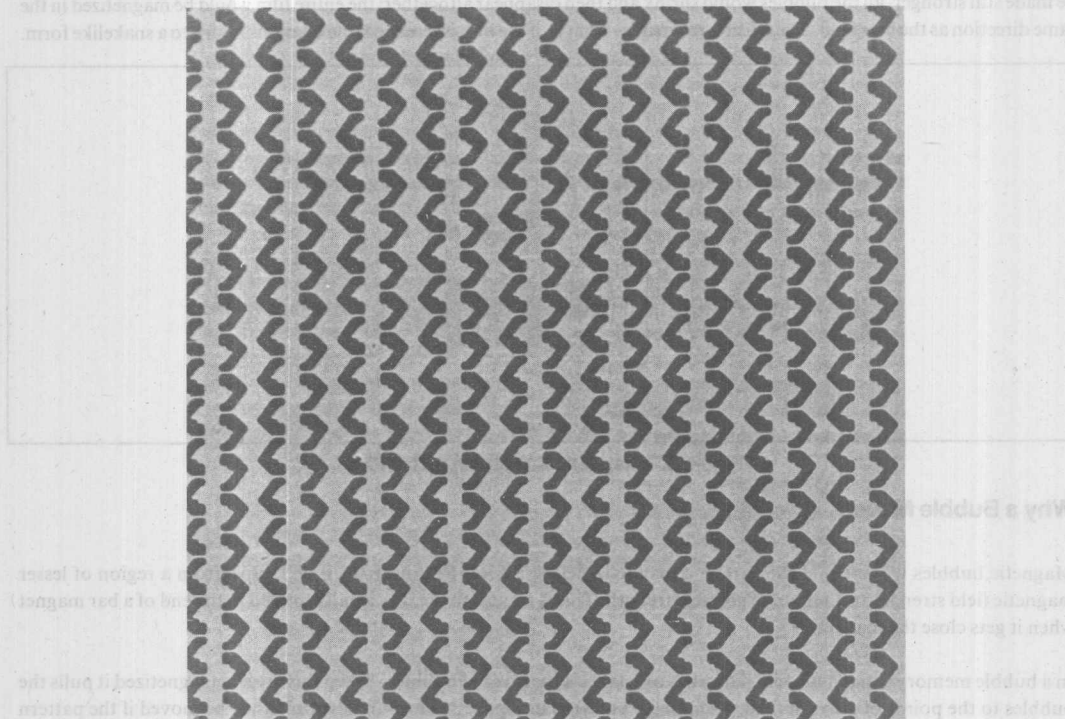


Photo 1. Asymmetric Chevrons Deposited on a Thin Film

Why Magnetic Bubbles are Non-Volatile

In a magnetic bubble memory system, the bias field in which the bubbles exist is generated by a pair of *permanent* magnets. The substrate bearing the thin film and its bubbles is mounted between these magnets and is therefore continuously subject to the bias field.

The rotating field that propels the bubbles through the film is generated by currents in two coils wrapped around the substrate at right angles to each other. These currents are generated by electronic circuits that are part of the magnetic bubble memory system. No mechanical motion is involved.

If power fails, the circuits stop operating, the rotating field disappears, and the bubbles stop moving. But the bias field, generated by the permanent magnet, is not affected. Therefore the bubbles and the data that they represent are maintained in the film. When power is restored the data is again accessible.

BUBBLE MEMORY MANUFACTURING TECHNOLOGY

Bubble memories are produced in a process that resembles semiconductor manufacturing in many ways (Figure 4). Manufacturing begins with a nonmagnetic garnet wafer on which a magnetic film is deposited, using conventional techniques. An ion implantation process alters the magnetization of the top surface of the film, discouraging the formation of abnormal bubbles with undesirable dynamic properties. Then nonmagnetic conductors, bubble-steering patterns of magnetic metal, insulation, passivation, and bonding pads are deposited in much the same way as successive layers on semiconductor integrated circuits. Patterns in each layer are defined photolithographically, just as with semiconductors.

Magnetic bubble technology differs from semiconductor technology in the materials used and in the complexity of the process. Semiconductor circuits use eight or more layers of silicon doped with various materials that affect its electrical characteristics, compared to about three layers of essentially pure metallic and insulating material in bubble technology. These materials are chosen for their magnetic rather than their electrical properties.

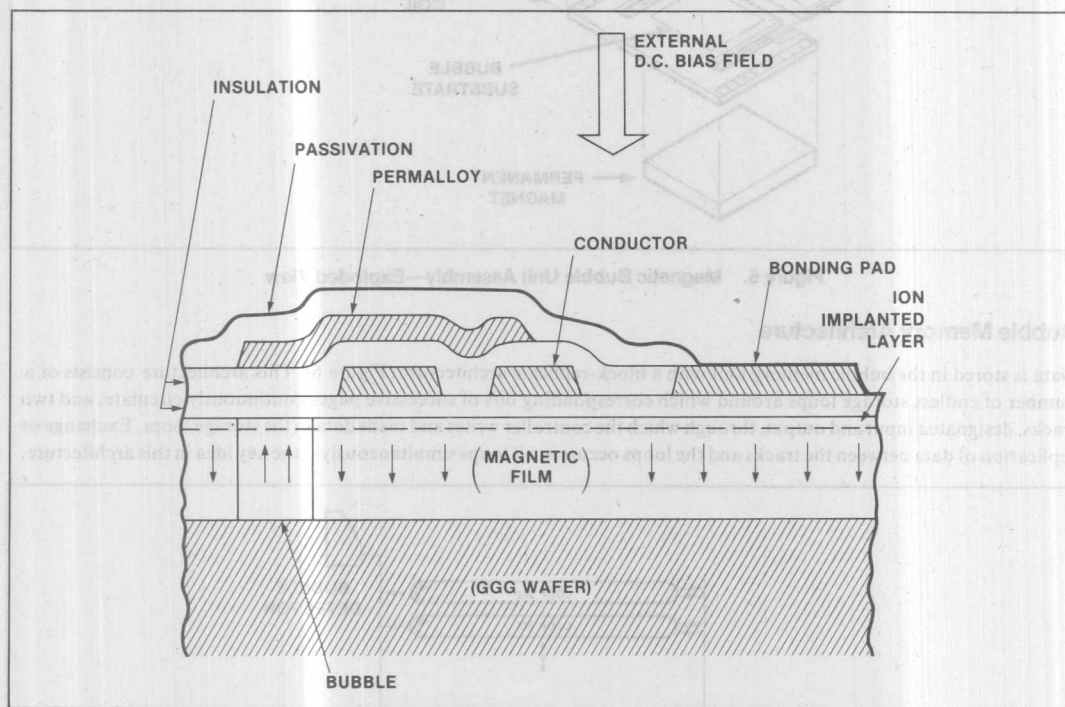


Figure 4. Magnetic Bubble Chip Cross Section

Bubble Memory Functional Description

The Intel 7110 magnetic bubble memory unit contains the bubble chip, the coils that generate the rotating field, two permanent magnets for the bias field, and a magnetic shield that prevents disturbances by external fields and forms a return path for the bias field around the bubble chip (Figure 5).

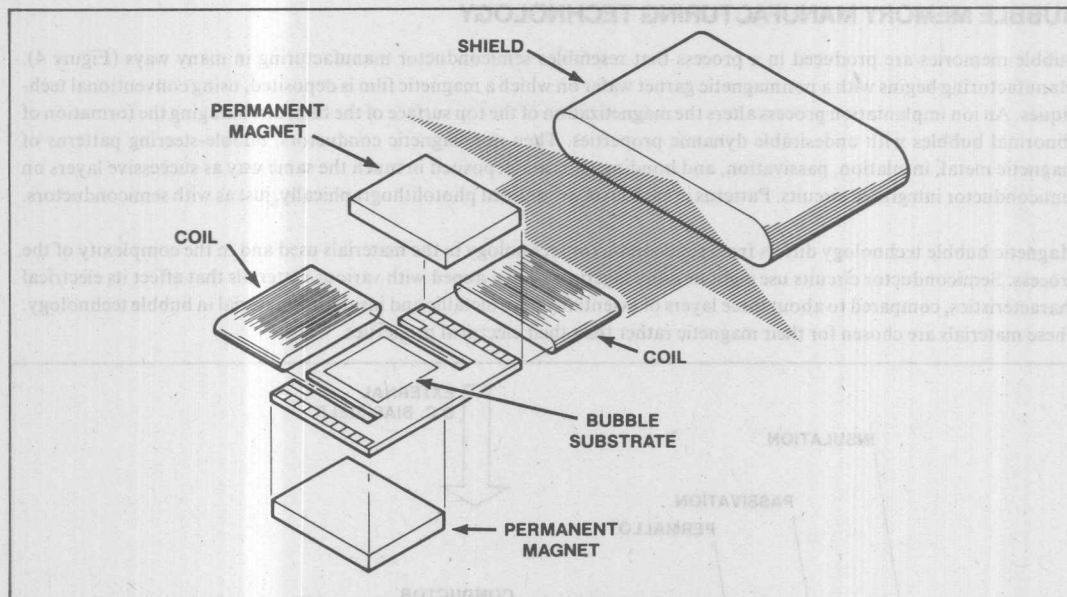


Figure 5. Magnetic Bubble Unit Assembly—Exploded View

Bubble Memory Architecture

Data is stored in the bubble memory unit with a block-replicate architecture (Figure 6). This architecture consists of a number of endless storage loops around which corresponding bits of successive pages continuously circulate, and two tracks, designated input and output, through which the controller writes and reads data in the storage loops. Exchange or replication of data between the tracks and the loops occurs in all loops simultaneously—the key idea in this architecture.

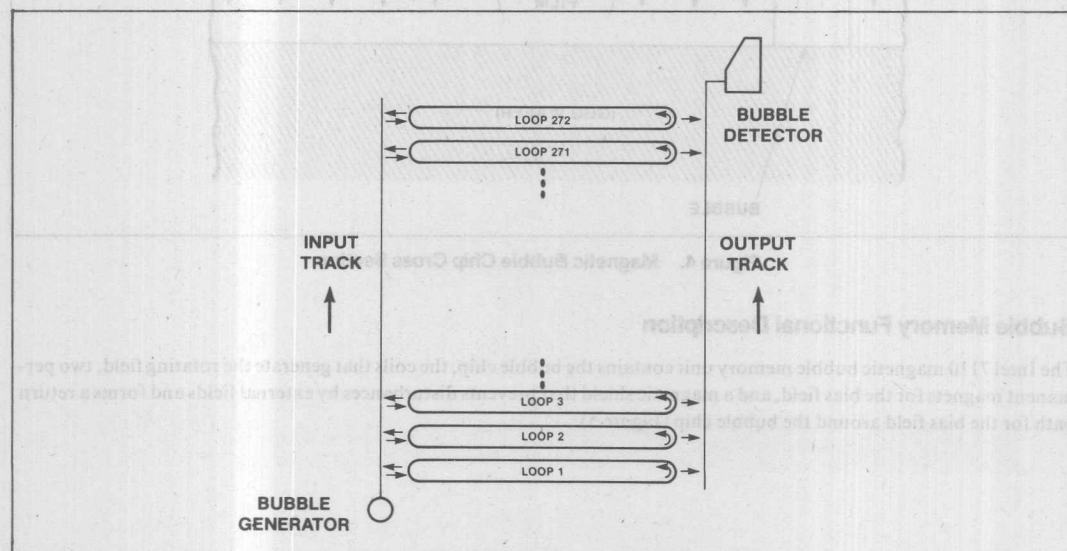


Figure 6. Block-Replicate Architecture

WRITING DATA INTO THE BUBBLE MEMORY

Seed Bubble

The seed bubble, at the beginning of the input track, is generated by an electric current pulse in a hairpin-shaped loop of conductive material. The pulse is strong enough to reverse the bias field locally and thus allow a bubble domain to be created. Once having been created, the seed bubble remains in existence as long as the external bias field is maintained.

The seed circulates under a permalloy patch, driven by the rotating field that propagates bubbles elsewhere in the memory. This bubble is constrained to a kidney shape by interaction of the bias and rotating field with the metal patch (Figure 7). The seed is split in two by a current pulse in the hairpin-shaped conductor. One of them remains under the patch as the seed, quickly regaining its original size; the other one, driven by the rotating field, is transferred to the input track section of the chip. The current pulse that splits the seed is generated to store a binary 1 in the memory; to store a 0, the pulse is omitted, and no bubble is generated.

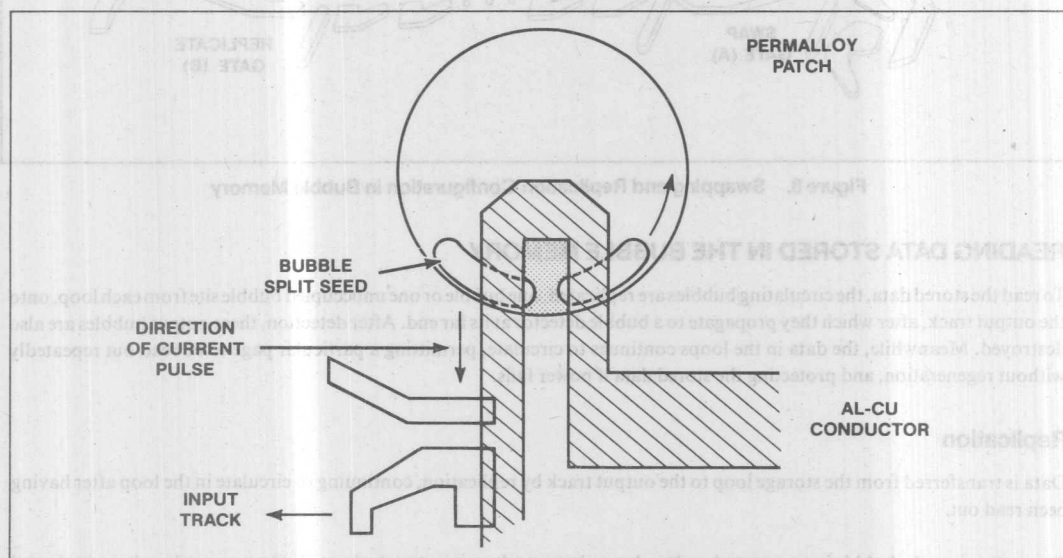


Figure 7. Seed Bubble and Bubble Generation

A seed bubble is maintained at one end of the input track. Bubbles corresponding to binary 1's in the input word are split from the seed and propagate along the input track. When the input track contains exactly one page (64 bytes) then the bubbles exchange places with old bubbles previously circulating in the loops. This is accomplished by an operation called swapping. Thereafter the new bubbles circulate, while the old bubbles now in the track propagate to the end and are destroyed.

Swapping

Transfer of data from the input track to a storage loop involves a swap, bringing the old data onto the input track for destruction at the end of the line, while the new data takes its place in the loop. This is done when a current pulse in an associated conductor under the chevrons causes a bubble to jump from the input track to the storage loop and vice versa. The swap pulse is essentially rectangular, preserving the bubble without cutting it in two.

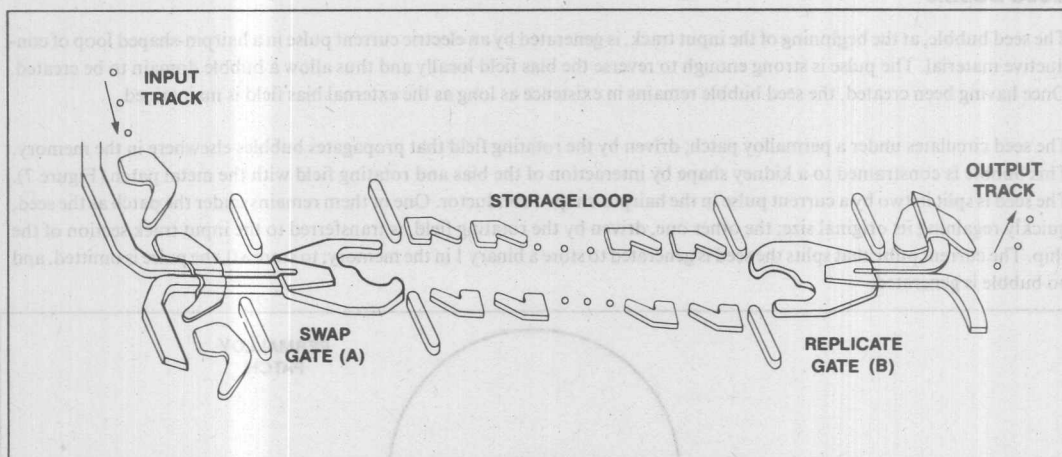


Figure 8. Swapping and Replication Configuration in Bubble Memory

READING DATA STORED IN THE BUBBLE MEMORY

To read the stored data, the circulating bubbles are replicated, one bubble or one unoccupied bubble site from each loop, onto the output track, after which they propagate to a bubble detector at its far end. After detection, these output bubbles are also destroyed. Meanwhile, the data in the loops continues to circulate, permitting a particular page to be read out repeatedly without regeneration, and protecting the stored data if power fails.

Replication

Data is transferred from the storage loop to the output track by replication, continuing to circulate in the loop after having been read out.

For replication, the bubble is propagated under a large element where it is stretched out. As it passes under a hairpin shaped conductor loop it is cut by a current pulse just as in bubble generation.

The replicating current pulse waveshape has a high, narrow leading spike for cutting the original bubble in two, and a lower and wider trailing portion during which the new bubble moves under the output track. The entire pulse lasts about one-quarter of a cycle of the rotating field. In this manner the data in the storage loops is replicated onto the output track, and yet retained in the storage loops in case of a sudden power failure.

Near the end of the output track is a bubble detector—essentially a magnetoresistive bridge formed by interconnecting the permalloy chevrons to make a continuous electrical path of maximum length (Figure 9). As bubbles pass under the bridge, the resistance changes slightly, modulating the currents through the bridge and creating an output voltage of several millivolts. Bubbles are stretched at right angles to the direction of propagation by adding parallel rows of chevrons; these stretched bubbles generate larger output signals at the detector. Beyond the detector, the output track runs the bubbles into the guard rail and destroys them.

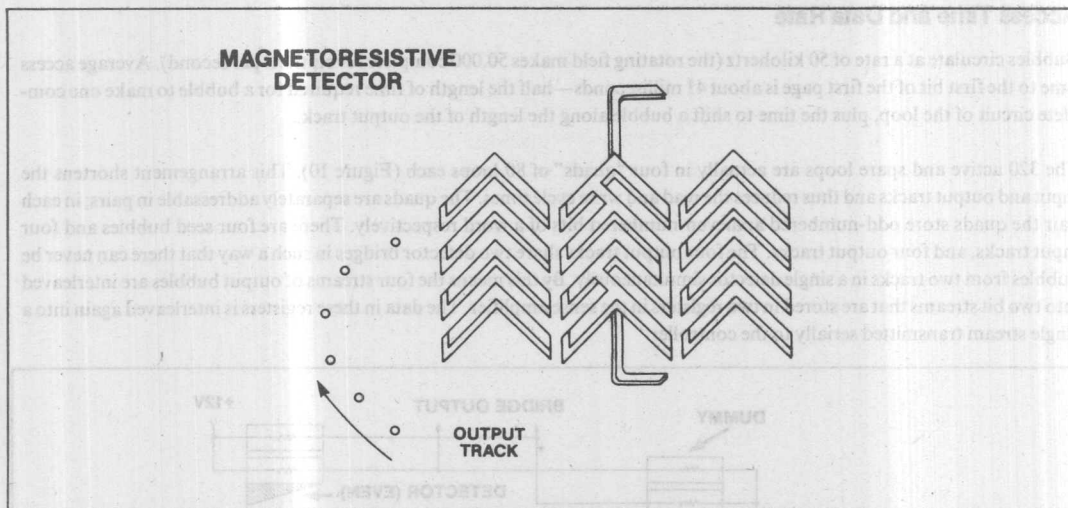


Figure 9. Bubble Detection

Redundancy

The Intel magnetic bubble memory unit physically stores data in 320 storage loops, with capacities of 4,096 bits each. Of the 320 loops, 272 are actually used (active) and 48 are spares (inactive); the boot loop records which loops are used.

Boot Loop

Some of the loops of an individual memory are set aside as spares. The decision as to which loops are to be used (active) and which are not to be used (inactive) is made after the memory unit has been assembled and is undergoing tests at the factory. The outcome of this decision is stored in an extra loop included in each memory chip, in the form of a 12 bit code for each "active" and "inactive" loop.

Whenever power is turned on in the memory system, the system must be initialized before it can be used. Part of the initialization process includes reading the contents of this extra loop, called the boot loop, and placing this information in a bootloop register in the formatter/sense amplifier. From then on, as long as power is on, this register identifies the "active" loops for both reading and writing; "inactive" loops are ignored. The formatter does not attempt to store data in "inactive" loops, and the sense amplifier ignores any data that appears from these loops.

Data Storage—External Appearance

Data is stored logically as 2,048 pages of 512 data bits each. 256 data bits plus 14 error-correction check bits and 2 unused bits are stored in each half of the bubble chip. If automatic error correction is not used, these 16 bits are available for data storage.

Error Correction

Error detection and correction can be performed in the formatter/sense amplifier, which includes a 14-bit cyclic redundancy code that corrects a single burst error of up to five bits in each 270-bit block including the code itself. These code bits are appended to the end of each 256-bit data block when writing into the cell, and checked when the block is read. The error correction feature can be used or not at the user's discretion, by properly setting a register in the bubble memory controller chip. If it is not used, the loops occupied by the code bits become available for additional data.

Access Time and Data Rate

Bubbles circulate at a rate of 50 kilohertz (the rotating field makes 50,000 complete revolutions per second). Average access time to the first bit of the first page is about 41 milliseconds—half the length of time required for a bubble to make one complete circuit of the loop, plus the time to shift a bubble along the length of the output track.

The 320 active and spare loops are actually in four “quads” of 80 loops each (Figure 10). This arrangement shortens the input and output tracks and thus reduces the read and write cycle times. The quads are separately addressable in pairs; in each pair the quads store odd-numbered and even-numbered bits of a word respectively. There are four seed bubbles and four input tracks, and four output tracks. The four output tracks share two detector bridges in such a way that there can never be bubbles from two tracks in a single detector simultaneously. By this means the four streams of output bubbles are interleaved into two bit streams that are stored in two registers in the sense amplifier. The data in these registers is interleaved again into a single stream transmitted serially to the controller.

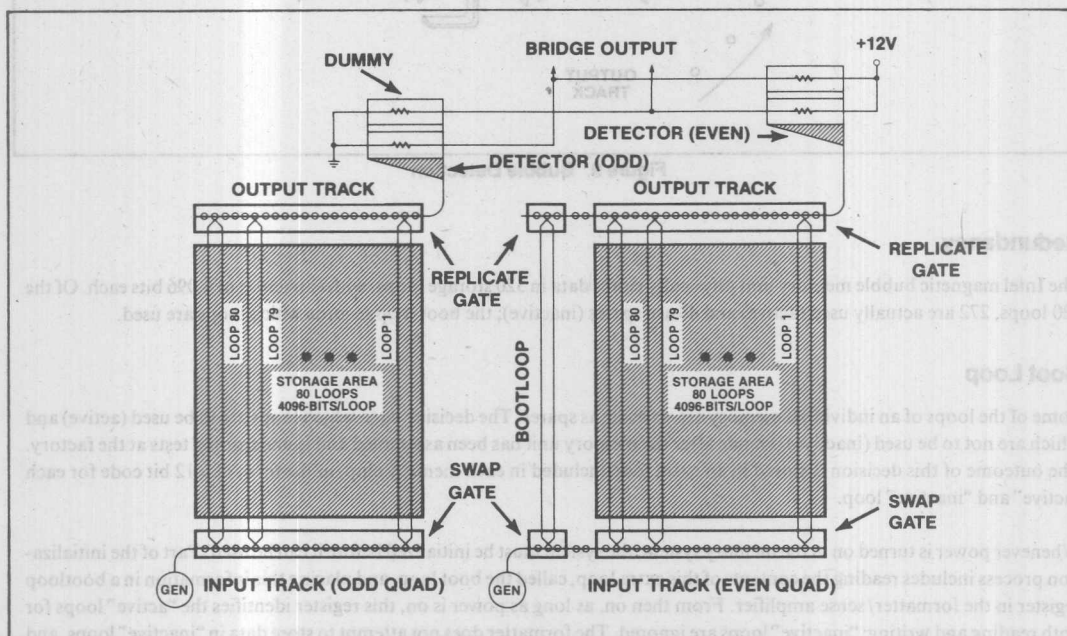


Figure 10. Organization of Bubble Memory (One-Half Chip)

SPECIFIC STRUCTURES OF A MAGNETIC BUBBLE MEMORY

A magnetic bubble memory system consists of a controller and up to eight 1-megabit magnetic bubble subsystems. A minimum system has a controller and one subsystem. The subsystem comprises one magnetic bubble unit in which the data is actually stored, and the peripheral units listed in Table 2 and diagrammed in Figure 11. These circuits are described later in this primer.

Table 2. Components of Intel Bubble Memory System

CONTROLLER	SUBSYSTEM
7220-1 Bubble Memory Controller (for 1 to 8 subsystems)	Memory 7110 Magnetic Bubble Unit
	Peripheral Units 7242 Formatter/Sense Amplifier 7230 Current Pulse Generator 7250 Coil Predriver 7254 Drive Transistor Assembly (2 required per subsystem)

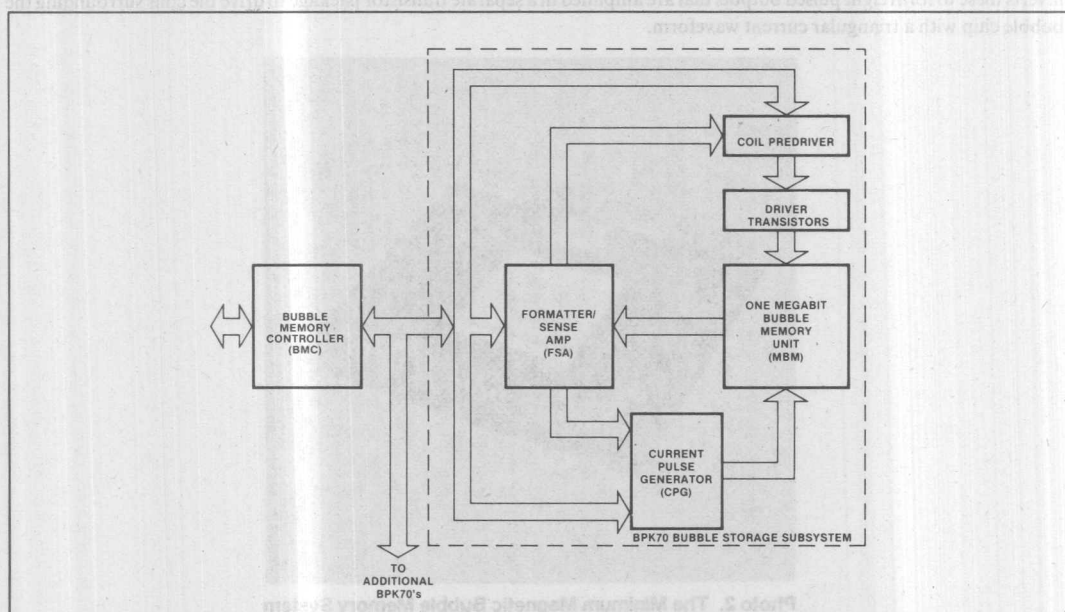


Figure 11. Minimum Magnetic Bubble Memory System, Shaded Portion is Bubble Subsystem

SUPPORT CHIPS

Five semiconductor integrated circuits are necessary to support each bubble chip. These components are described in some detail in the following paragraphs. In addition, each bubble memory system requires a controller, a separate integrated circuit described later.

Formatter/Sense Amplifier (FSA)

Serial data to be stored in or read from the bubble memory passes through the FSA. The FSA keeps track of which loops in the bubble memory are spares, executes the error correction coding and decoding if it is implemented, and shifts data to the bubble memory input tracks or from the output tracks, amplifying the output signals from the memory.

The FSA has a chip-select input, which is normally grounded (permanently enabled). However, each FSA drives the chip-select input of other circuits associated with the same bubble chip, so they are all enabled at the proper time.

Current Pulse Generator (CPG)

All signals except those that control the rotating field originate in the CPG. This device is the source of a current pulse that cuts a new bubble from the seed bubble whenever the FSA has a binary 1 to be stored. Later, when this bubble reaches the loop in which it is to reside, the CPG issues the signal that swaps it with the bubble or non-bubble previously stored in that location of the loop. When data is to be read, the bubble is replicated on the output track by still another signal from the CPG.

Coil Predriver (CPD)

Four digital signals (positive and negative versions of both X and Y waveforms) are sent to the CPD from the controller with appropriate durations and phases to control the rotating field that moves the bubbles in the memory. The CPD combines and inverts these to form eight pulsed outputs that are amplified in a separate transistor package to drive the coils surrounding the bubble chip with a triangular current waveform.

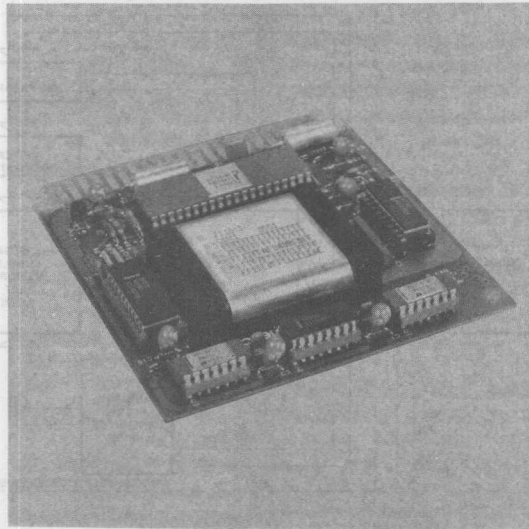


Photo 2. The Minimum Magnetic Bubble Memory System Including Controller

CONTROLLER

The bubble memory controller is the interface between the memory system and the equipment it serves. It converts serial data to parallel and parallel data to serial, and generates all timing signals required by the other support circuits in the bubble memory system. It can control up to eight bubble subsystems, for a total of a megabyte of memory.

Internal storage on the controller includes a first-in-first-out buffer with a capacity of 40 bytes. This buffer stores data to be sent serially to the FSA or just received from the FSA on one side, and data to or from the parallel bus served by the bubble memory on the other. It also serves as a speed matching device between the user at the parallel bus and the FSA which must transfer data to and from the bubble device at exactly the rotating field ratio in each channel.

Bias field—a magnetic field perpendicular to a magnetic thin film that maintains conditions necessary to support formation of magnetic bubbles in the film.

Boot loop—in a magnetic bubble memory with serial/parallel/serial architecture and redundant loops, a special loop containing information that identifies which loops are active and which are inactive, as determined by factory test. This loop also contains the information necessary to synchronize the bubble memory page locations with the controller after power up.

Bubble, magnetic—a cylindrical magnetic domain in a thin film of orthoferrite or garnet. When viewed from above, the cylindrical shape appears spherical, hence the name "bubble." A bubble represents a binary 1 in most magnetic bubble memories.

Chevron—one of many possible shapes for a magnetic pattern deposited on a thin film to steer bubbles in a desired direction. Asymmetric chevrons are used in Intel memories.

Detector—a means of distinguishing bubbles from non-bubbles (1s from 0s) when a word is read from the bubble memory.

Domain, magnetic—a small region of a ferromagnetic substance that contains many similarly oriented atoms, so that the region as a whole is magnetized in that direction.

E²PROM—an acronym for electrically erasable programmable read-only memory, which is a memory component that, though nominally read-only, can accept changes to any work stored in it by electrical means, but at substantially slower speed than that at which stored words are read.

EPROM—an acronym for erasable programmable read-only memory, which is a memory component that, though nominally read-only, can be completely erased, usually by exposure to ultraviolet light, and then reloaded with new information, but at substantially slower speed than that at which stored words are read.

Ferrite—any of several compounds of iron, oxygen, and another metal, with magnetic properties that are useful in certain microwave applications and in computer memories.

Field, magnetic—a region of space in which a magnetic force exists and can be measured.

Garnet—a naturally occurring silicate mineral sometimes used in jewelry. Synthetic garnets with the same crystal structure can be made of oxides of iron and yttrium or one of the rare earths. Garnet is the preferred material for the thin magnetic film in a bubble memory.

Input track—a series of magnetic metal patterns that control the movement of bubbles in a thin film, and thereby lead them from a bubble generator toward one or more storage patterns.

Ion implantation—a process involving accelerators, similar to the machines used by nuclear physicists, for depositing dopants on and just below the surface of an electronic component; used to alter the physical properties of the material.

Latency—a delay between a request to read or write data in a memory and the actual beginning of the operation, imposed by a requirement for the address to move physically (but not necessarily mechanically) to a point where the data transfer can take place.

Magnetization vector—an expression of the magnitude and direction of a magnetic field at a point in space.

Magnetoresistance—a change in electrical resistance due to the presence of a magnetic field.

Major loop—in a magnetic bubble memory, an endless loop containing a bubble generator, a bubble detector, and/or a bubble annihilator, through which data is read or written, and which transfers bubbles to or from one or more minor loops (q.v.) in which they are stored. In some designs the major loop is not endless, and all bubbles not transferred out of it collapse when they reach the end. In these cases the major loop becomes an input or output track (q.v.).

Minor loop—in a magnetic bubble memory, an endless loop in which bubbles are stored, having been transferred into it from a major loop or input track (q.v.) and accessible by transfer into a major loop or output track (q.v.).

Non-Volatility—a property of some memory technologies that retains the integrity of stored data when power is turned off.

Orthoferrite—one of several oxides of iron and either yttrium or a rare earth. The molecular structure is simpler than that of garnet (q.v.). Orthoferrites were the first materials used for the thin magnetic film in experimental bubble memories, but have yielded to garnets, which have more desirable properties—notably ease of preparation as thin films with the necessary magnetic characteristics.

Output track—a series of magnetic metal patterns that control the movement of bubbles in a thin film, and thereby lead them from one or more storage patterns toward a bubble detector.

Permalloy—an easily magnetized and demagnetized alloy of nickel and iron.

PROM—acronym for programmable read-only memory—a read-only memory whose content is loaded by the user after delivery, as opposed to read-only memories whose content is fixed during manufacture. Once loaded, the data in a PROM is not alterable.

Pseudo-random access—a property of some memory technologies in which the time of access to blocks of stored data is largely (but not necessarily entirely) independent of the position of the block in the storage medium, but in which the time of access to bits, words or other entities depends on the position of that entity within the block.

Random access—a property of some memory technologies in which the time of access to any stored bit, word, or other entity is wholly independent of that entity's position in the storage medium.

Saturation—a state of magnetization of a material by a field such that, if the field is increased, the magnetization of the material does not increase and the magnetic flux density increases in proportion to the field (having increased much more rapidly in weaker fields).

Seed—a permanent bubble in a magnetic bubble memory, from which other bubbles are cut to represent stored binary 1s.

Serial access—a property of some memory technologies in which the time of access to any stored bit, word, or other entity depends strongly on that entity's position in the storage medium.

Thin film—any film of material deposited on a suitable substrate to take advantage of the material's special properties when dispersed as a film. Thickness ranges usually from about 10^{-9} to 10^{-6} meter, and occasionally to 10^{-5} meter or more, as in bubble memories.

T-I bar—one of several possible shapes for a magnetic pattern deposited on a thin film to steer bubbles in a desired direction, consisting of shapes like the letter T and the letter I alternately along a track. This pattern was used extensively in early bubble memory designs, but is no longer generally employed.



iSBC™ 464

64K BYTE EPROM EXPANSION BOARD

- Provides EPROM/ROM expansion of iSBC™ 80, iSBC™ 86 and iSBC™ 88 systems via direct MULTIBUS interface
- Sockets for up to 64K bytes of EPROM
- Compatible with Intel® 2758, 2716 or 2732/2732A erasable PROMs

- Switch selectable base address on 4K byte boundaries for each memory bank
- Assignable anywhere within a 1 megabyte address space
- EPROM components which are not enabled are placed in standby power mode
- Requires a single +5V power supply

The iSBC 464 is a member of Intel's complete line of iSBC memory and I/O expansion boards. The iSBC 464 board interfaces directly to the iSBC 80, iSBC 86 or iSBC 88 single board computers via the MULTIBUS system bus, to expand system EPROM memory capacity.

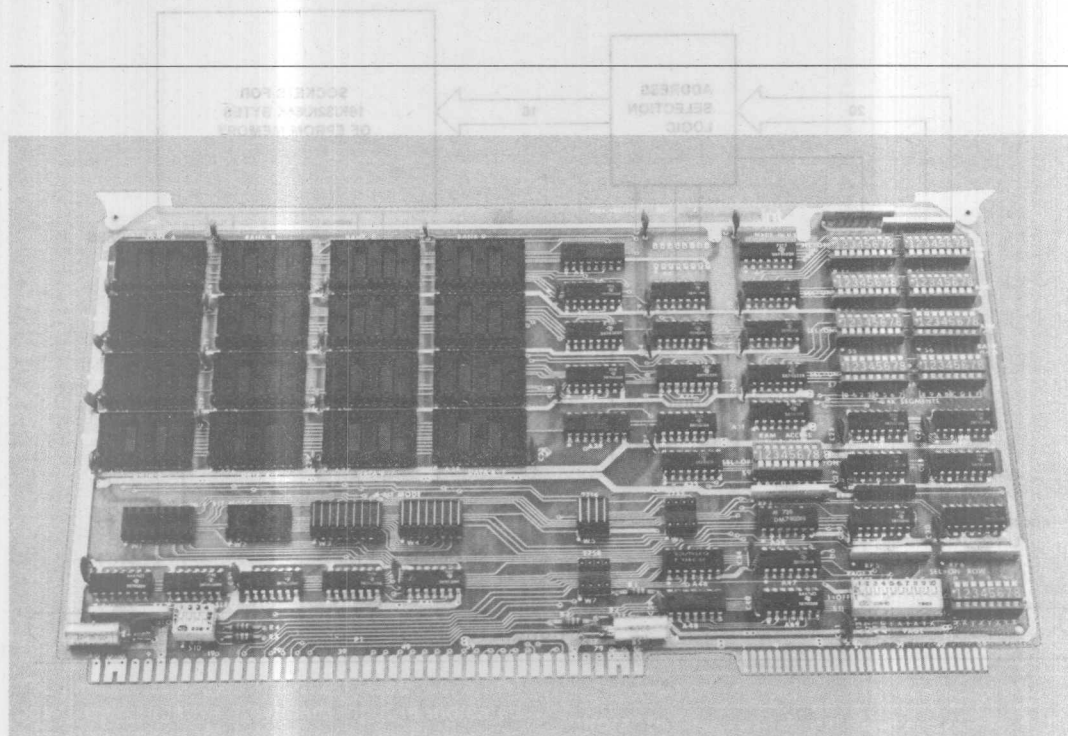


Figure 1. iSBC 464 Block Diagram

FUNCTIONAL DESCRIPTION

Memory Configuration

The iSBC 464 board contains sixteen sockets which provide a maximum of 64K bytes of memory expansion. The actual capacity of the board is determined by the type and quantity of EPROM components installed by the user. The board is compatible with three different sizes of Intel EPROM devices. These are the 1K byte 2758 EPROM, the 2K byte 2716 EPROM, and the 4K byte 2732 EPROM.

Mode of Operation — The iSBC 464 board can operate in one of two modes: the 8 bit only mode or the 16/8 bit mode. The 8 bit mode provides the most efficient memory configuration for systems handling 8 bit data. The 16/8 bit mode allows 16 bit words to be accessed by 16 bit processors. In the 16/8 bit mode, 16 bit and 8 bit microprocessors may also access either the high order byte or the low order byte of a 16 bit word. The mode of operation is selected by placing two option jumper blocks in the appropriate sockets.

Memory Banks — When used in the 8 bit mode, the iSBC 464 board is organized into four banks (labeled A-D) of four sockets each. Depending on the type of memory components used, each bank may contain a maximum of 4K, 8K or 16K bytes of memory. Unused memory sockets may be deselected by bank or individually in bank D. Deselecting a bank or individual socket frees that address space for use elsewhere in the system. In the 16/8 bit mode, banks A & B and C & D are paired together to form two banks (labeled AB, CD) which are 16 bits wide. Each of these banks has four socket pairs. Bank AB may be deselected as a single unit. Socket pairs in bank CD may be deselected individually. Thus, board configurations using fewer than 16 memory components do not fill memory address space with unused sockets. Selection/deselection is accomplished by setting switches on the board.

Memory Access Time — The iSBC 464 board operates with one of 15 switch selectable memory access times ranging from 35 to 1435 nanoseconds. This feature allows the board to be tailored to the performance of the installed components and the system CPU.

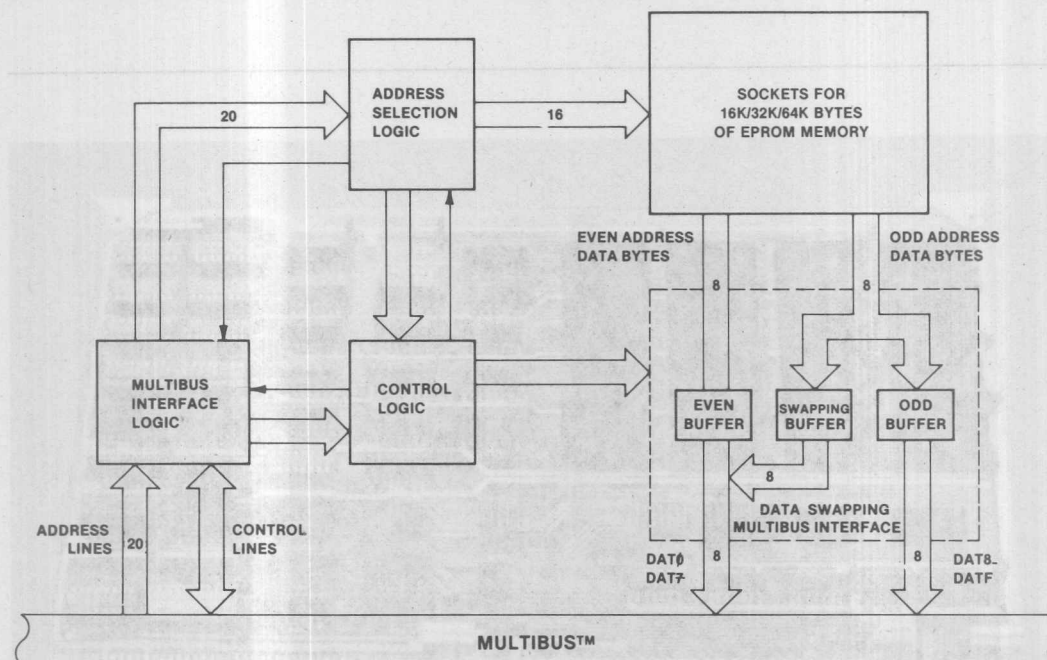


Figure 1. iSBC 464 Block Diagram

Memory Addresses

Switch selectable options on the iSBC 464 board allow the board to be assigned anywhere within a 1 megabyte address space. In either operating mode, the base address of each memory bank may be set to any 4K byte boundary within a 64K byte memory page. There is one exception. If the 4K byte devices are used in the 16/8 bit mode, then base addresses are restricted to 8K byte boundaries. If the board is used in a system with an address range greater than 64K bytes, memory on the iSBC 464 board may reside in one or two 64K byte memory pages. Any two pages out of a possible 16 may be chosen by setting switches on the board.

Standby Power Operation

The iSBC 464 board takes advantage of the standby modes of the Intel 2758, 2716 and 2732. When they are not enabled, these components draw as little as 25% of

their active level power with no degradation in access time. The iSBC 464 board is designed so that only two memory components are enabled during a read operation.

RAM Overlap

Memory banks of the iSBC 464 board can be overlapped with the addresses of system RAM by setting on-board switches. The process of addressing a memory bank will drive the Inhibit RAM (INH1/) signal true. This signal is issued to the MULTIBUS system bus in order to prevent any MULTIBUS accessible RAM in the system from responding to the current address. If an EPROM is addressed which has its corresponding RAM overlap switch on, an access time of 15 clock cycles is imposed. This allows overlapped dynamic RAM to refresh before the address on the MULTIBUS is changed. The RAM overlap feature does not apply to RAM which is not on the MULTIBUS system bus.

SPECIFICATIONS

Word Size

8 bits or 16 and 8 bits

Memory Size

Sockets are provided for up to 16K bytes in 1K increments or 32K bytes in 2K increments or 64K bytes in 4K increments

Compatible Intel® Memory

EPROM — 2758 or 2716 or 2732

INTERFACE — All 20 address, 16 data, and 6 control signals are TTL compatible and Intel MULTIBUS compatible

Electrical Characteristics

DC Power (max)

V_{CC}: +5V DC $\pm 5\%$

I_{CC}: 1.1 amps without EPROMs

I_{CC}: 1.6 amps with (16) 2716s or 2758s

I_{CC}: 1.3 amps with (16) 2732s or 2732As

Connectors

Bus — 86-pin double-sided PC edge connector with 0.40 cm (0.156 in.) contact centers

Mating Connector — Viking 3KH43/9AMK12 or compatible connector

Physical Characteristics

Length — 30.48 cm (12 in.)

Height — 17.15 cm (6.75 in.)

Depth — 1.27 cm (0.5 in.)

Weight — 294 gm (10.5 oz) without EPROM

Environment

Operating Temperature — 0°C to +55°C

Relative Humidity Limits — <90% non-condensing

Reference Manual

9800643A — iSBC 464 Memory Expansion Board Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 464	64K EPROM Expansion Board

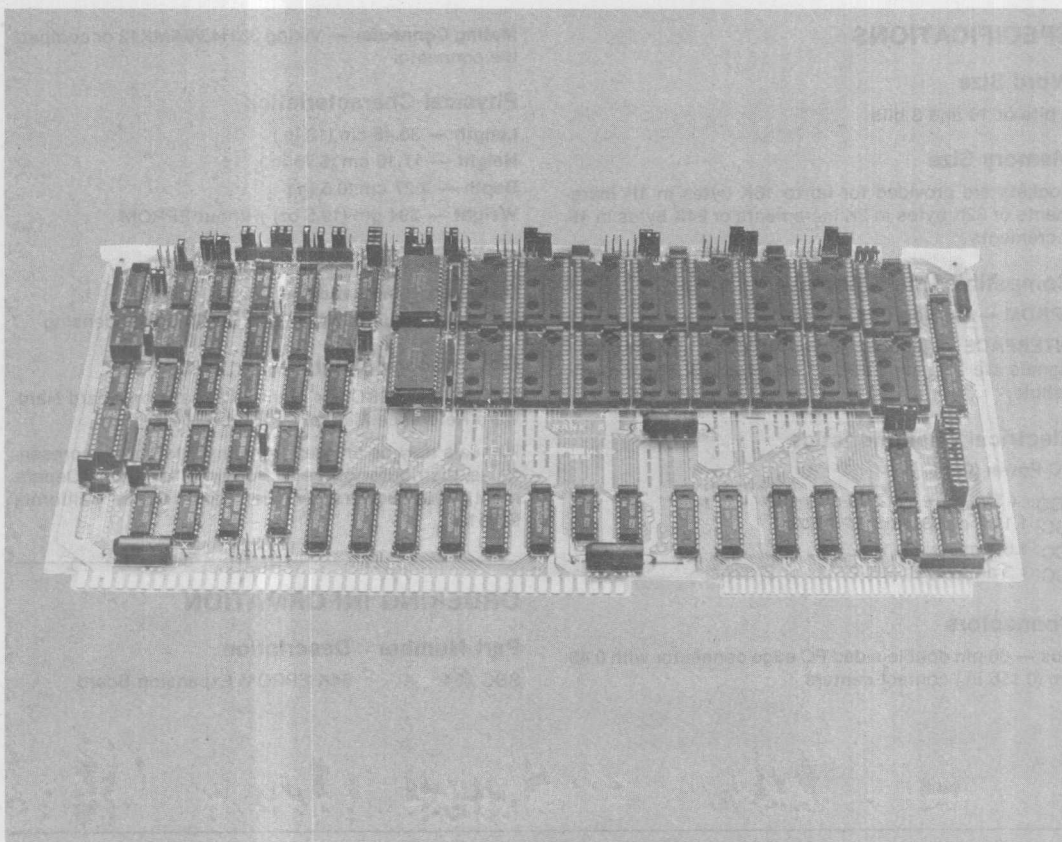


PRELIMINARY

iSBC® 428 UNIVERSAL SITE MEMORY EXPANSION BOARD

- Supports EPROM, ROM, E²PROM, SRAM, IRAM and NVRAM
- iLBX™ BUS or MULTIBUS® Selectable
- Provides support for Battery Backup/Memory Protect
- Sixteen 28 pin Universal sites
- Assignable anywhere within a 16 megabyte address space on 256K byte boundaries
- Jumper selectable base address on 4K byte boundaries

The iSBC® 428 Universal Site Board is a member of Intel's complete line of Memory and I/O Expansion boards. The iSBC 428 Universal Site Memory Expansion Board interfaces directly to the iSBC 80, iSBC 88, or iSBC 86 Single Board Computers via the MULTIBUS® System Bus to expand system memory requirements, while system memory expansion requirements for iSBC 286 Single Board Computer can interface via either the MULTIBUS or the high speed iLBX™ Bus.

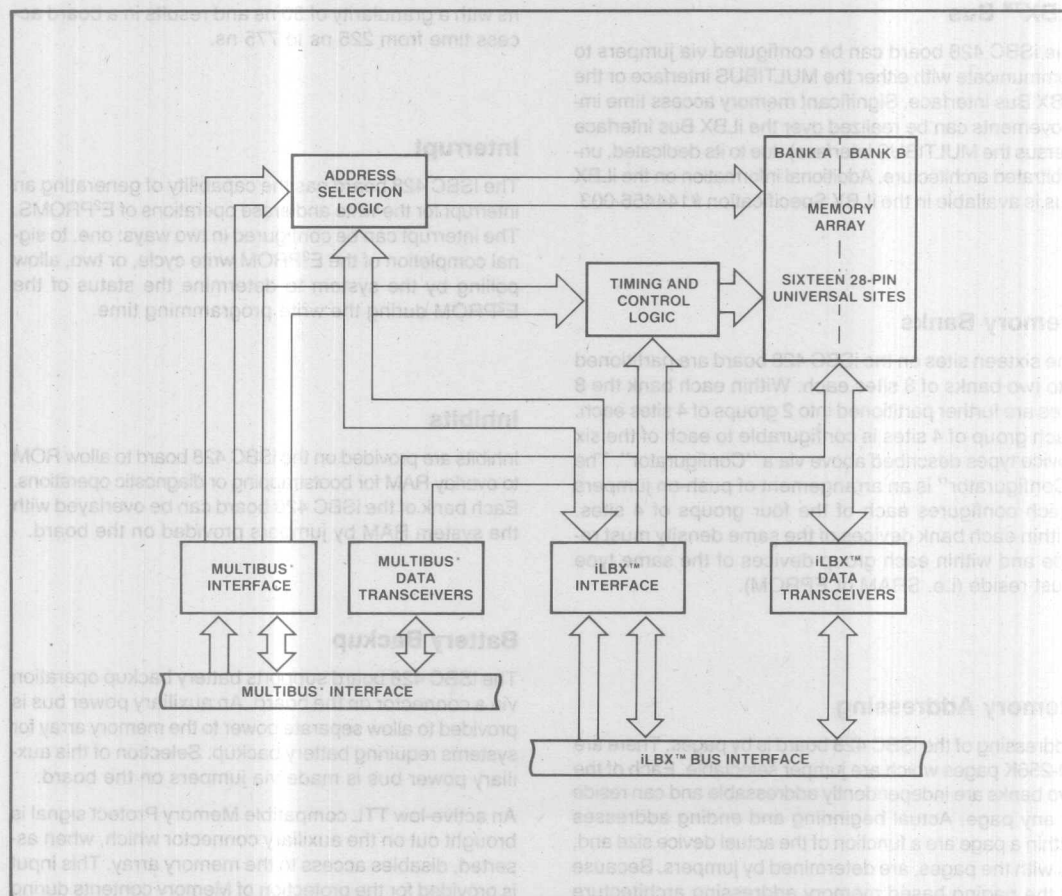


Devices Supported

Listed below are the current and future devices supported by the iSBC 428 board.

Type	Size							Comments
	512 × 8	2K × 8	4K × 8	8K × 8	16K × 8	32K × 8	64K × 8	
EPROM	—	2716	2732A	2764	27128	27256	X	—
ROM	—	X	X	X	X	X	X	—
EEPROM	—	2817A	X	X	X	X	—	5V, Enhanced
SRAM	—	X	X	X	X	X	—	NMOS & CMOS
NVRAM	—	X	X	X	—	—	—	—
IRAM	—	—	—	2186	—	X	—	—

X-Denotes that the iSBC 428 board will support the device indicated but that it is not currently available from Intel.



iSBC® 428 Block Diagram

FUNCTIONAL DESCRIPTION

General

The iSBC 428 board contains sixteen 28 pin sockets. The actual capacity of the board is determined by the type and quantity of components installed by the user. The iSBC 428 board is compatible with five different types and densities of devices: the 2K by 8 thru 64K by 8 EPROM/ROM devices, 2K by 8 thru 8K by 8 "Five Volt Only, Enhanced" E²PROM devices, 512 by 8 thru 16K by 8 NVRAM (Non-Volatile RAM) devices, 2K by 8 thru 32K by 8 SRAM devices, and 8K by 8 IRAM (Integrated RAM) devices. In addition the board can be accessed by either the MULTIBUS System Bus or Intel's new high speed iLBX Bus.

iLBX™ Bus

The iSBC 428 board can be configured via jumpers to communicate with either the MULTIBUS interface or the iLBX Bus interface. Significant memory access time improvements can be realized over the iLBX Bus interface (versus the MULTIBUS interface) due to its dedicated, un-arbitrated architecture. Additional information on the iLBX Bus is available in the iLBX Specification #144456-003.

Memory Banks

The sixteen sites on the iSBC 428 board are partitioned into two banks of 8 sites each. Within each bank the 8 sites are further partitioned into 2 groups of 4 sites each. Each group of 4 sites is configurable to each of the six device types described above via a "Configurator". The "Configurator" is an arrangement of push-on jumpers which configures each of the four groups of 4 sites. Within each bank devices of the same density must reside and within each group devices of the same type must reside (i.e. SRAM or EPROM).

Memory Addressing

Addressing of the iSBC 428 board is by pages. There are 64-256K pages which are jumper selectable. Each of the two banks are independently addressable and can reside in any page. Actual beginning and ending addresses within a page are a function of the actual device size and, as with the pages, are determined by jumpers. Because of the paging based memory addressing architecture more than one iSBC 428 board can be placed in a system.

Mode of Operation

The iSBC 428 board can operate in one of two modes: the 8 bit only mode or the 8/16 bit mode. The 8 bit mode provides the most efficient memory configuration for systems handling 8 bit data only. The 8/16 bit mode allows the iSBC 428 board to be compatible with systems employing 8 bit and 16 bit masters. The mode of operation is selected by on board jumpers and is available for both MULTIBUS and iLBX Bus configurations.

Memory Access

The iSBC 428 board has jumper selectable access times which allows the board to be tailored to the performance of the particular devices which are installed in the iSBC 428 board. The board can be configured via jumpers to accept devices with an access time range of 50 ns to 500 ns with a granularity of 50 ns and results in a board access time from 225 ns to 775 ns.

Interrupt

The iSBC 428 board has the capability of generating an interrupt for the write and erase operations of E²PROMS. The interrupt can be configured in two ways: one, to signal completion of the E²PROM write cycle, or two, allow polling by the system to determine the status of the E²PROM during the write programming time.

Inhibits

Inhibits are provided on the iSBC 428 board to allow ROM to overlay RAM for bootstrapping or diagnostic operations. Each bank of the iSBC 428 board can be overlayed with the system RAM by jumpers provided on the board.

Battery Backup

The iSBC 428 board supports battery backup operation via a connector on the board. An auxiliary power bus is provided to allow separate power to the memory array for systems requiring battery backup. Selection of this auxiliary power bus is made via jumpers on the board.

An active-low TTL compatible Memory Protect signal is brought out on the auxiliary connector which, when asserted, disables access to the memory array. This input is provided for the protection of Memory contents during system power-down sequences.

SPECIFICATIONS

Word Size

8 or 8/16 bits

Memory Size

Sockets are provided for up-to sixteen 28 pin devices which can provide up to 512K bytes of EPROM/ROM/ SRAM.

Access Time

Jumperable from 225 to 775 ns with a granularity of 50 ns and is equivalent for both MULTIBUS and the iLBX Bus.

Power Requirements

$V_{CC} = 5 \text{ volts} \pm 5\%$

$I_{CC} = 2.0 \text{ amps}$, maximum, without any memory devices in the board.

Physical Characteristics

Length — 30.48 cm (12 inches)

Width — 17.15 cm (7.05 inches)

Depth — 1.27 cm (0.5 inches)

Environment

Operating Temperature — 0°C to +55°C

Relative Humidity — 90% non-condensing

Reference Manual

145696-001 — iSBC 428 Hardware Reference Manual (NOT SUPPLIED)

Additional Literature

9800683-04 — MULTIBUS Specification

144456-001 — The iLBX Specification

ORDERING INFORMATION

Part Number Description

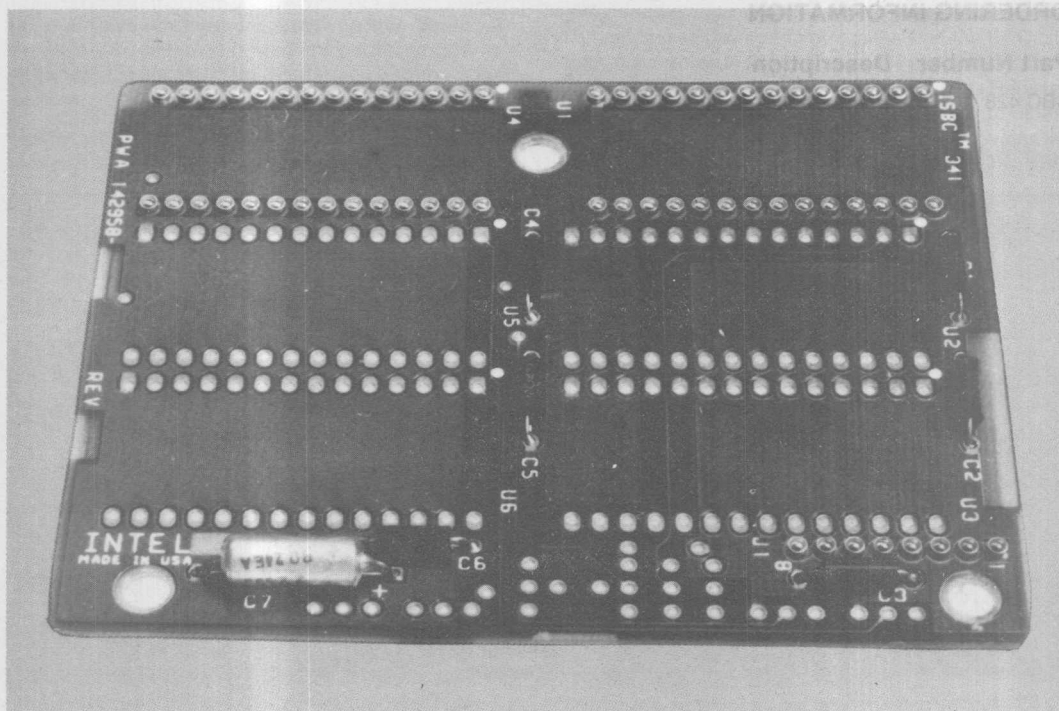
SBC 428	Universal Site Memory Expansion Board
---------	---------------------------------------

iSBC 341 28-PIN MULTIMODULE EPROM

- On-board memory expansion for iSBC 86/05, iSBC 88/25, and iSBC 88/40 microcomputers
- Supports JEDEC 24/28-pin standard memory devices, including EPROMs, byte-wide RAMs, and E²PROMs
- Sockets for up to 64K bytes of expansion with Intel® 27128 EPROMs
- On-board expansion provides “no wait state” memory access with selected devices
- Simple, reliable mechanical and electrical interface

The iSBC 341 28-pin MULTIMODULE EPROM board provides simple, low-cost expansion of the on-board EPROM capacity of the iSBC 86/05 Single Board Computer, the iSBC 88/25 Single Board Computer and the iSBC 88/40 Measurement and Control Computer. Four additional 28-pin sockets support JEDEC 24/28-pin standard devices, including EPROMs, byte-wide static and pseudo-static RAMs.

The MULTIMODULE expansion concept provides the optimum mechanism for incremental memory expansion. Mounting directly on the microcomputer, the benefits include low cost, no additional power requirements beyond the memory devices, and higher performance than MULTIBUS-based memory expansion.



FUNCTIONAL DESCRIPTION

The iSBC 341 28-pin MULTIMODULE EPROM option effectively doubles the number of sockets available for EPROM on the base microcomputer board on which it is mounted. The iSBC 341 board contains six 28-pin sockets. Two of the sockets have extended pins which mate with two of the sockets on the base board. Two of the EPROMs which would have been inserted in the base board

are then reinserted in the iSBC 341 sockets. Additional interface pins also connect chip select lines and power. The mechanical integrity of the assembly is assured with nylon hardware securing the unit in two places.

Through its unique interface, the iSBC 341 board can support 8 or 16-bit data paths. The data path width is determined by the base board — being 8 bits for the iSBC 88/40 and iSBC 88/25 microcomputers, and 8/16 bits for the iSBC 86/05 board.

SPECIFICATIONS

Word Size

8 or 8/16 bits (determined by data path width of base board).

Memory Size

32K bytes with available technology (JEDEC standard defines device pin-out to 128K-bit devices).

Device Size (Bytes)	EPROM Type	Max. iSBC 341 Capacity (Bytes)
2K × 8	2716	8K
4K × 8	2732A	16K
8K × 8	2764	32K
16K × 8	27128	64K

Access Time

Varies according to base board and memory device access time. Consult data sheet of base board for details.

Memory Addressing

Consult data sheet of base board for addressing data.

POWER REQUIREMENTS

Devices ¹	Max. Current @ 5V ± 5%
2716	420 mA
2732A	600 mA
2764	600 mA

NOTE:

1. Incremental power drawn from host board for four additional devices.

Auxiliary Power

There are no provisions for auxiliary power (battery backup) on the iSBC 341 option.

Physical Characteristics

WIDTH — 3.4 in. (8.64 cm)

LENGTH — 2.7 in. (6.86 cm)

HEIGHT — 0.78 in. (1.98 cm) *

WEIGHT — 5 oz (141.5 gm)

*Includes height of mounted memory devices and base board.

All necessary mounting hardware (nylon screws, spacers, nuts) is supplied with each kit.

Environmental Characteristics

OPERATING TEMPERATURE — 0°C to +55°C

RELATIVE HUMIDITY — to 90% (without condensation)

Reference Manuals

All necessary documentation for the iSBC 341 module is included in the CPU board Hardware Reference Manuals (NOT SUPPLIED)

iSBC 86/05 — Order No. 143153-001

iSBC 88/25 — Order No. 143825-001

iSBC 88/40 — Order No. 124978-001

Manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

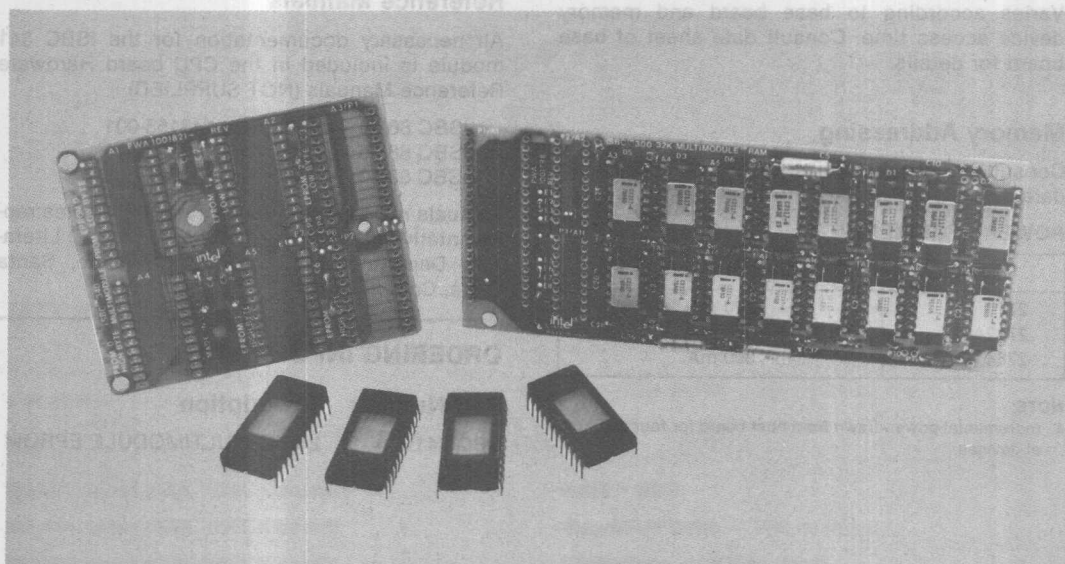
SBC 341 28-Pin MULTIMODULE EPROM



iSBC 300 or (pSBC 300*) 32K-BYTE RAM EXPANSION MODULE iSBC 340 or (pSBC 340*) 16K-BYTE EPROM EXPANSION MODULE

- On-board memory expansion for iSBC 86/12A Single Board Computer
- iSBC 300 module provides 32K bytes of dual port dynamic RAM and plugs directly into the iSBC 86/12A board
- iSBC 340 module provides sockets for up to 16K bytes of additional EPROM and plugs directly into the iSBC 86/12A board
- On-board memory expansion eliminates MULTIBUS system bus latency and increases system throughput
- Low power requirements
- Simple, reliable mechanical and electrical interconnection

The iSBC 300 32K-byte RAM expansion module and the iSBC 340 16K-byte EPROM expansion module provide simple, low cost expansion of the memory complement available on the iSBC 86/12A single board computer. Each module utilized individually or together can double the iSBC 86/12A board's on-board RAM and EPROM memory capacity. The iSBC 300 32K-byte RAM expansion module and the iSBC 340 16K-byte EPROM expansion module options for the iSBC 86/12A board offer system designers a new level of flexibility in defining and implementing Intel® single board computer systems. These options allow the systems designer to double the memory complement of an iSBC 86/12A board with a minimum of system implications. Because they expand the memory configuration on-board, they can be accessed as quickly as the existing iSBC 86/12A memory by eliminating the need for accessing the additional memory via the MULTIBUS system bus. With the iSBC 86/12A board mounted in the top slot of an iSBC 604 or iSBC 614 cardcage, sufficient clearance exists for mounting both the iSBC 300 and/or the iSBC 340 expansion module option(s). If the iSBC 86/12A board is inserted into some other slot, the combination of boards will physically (but not electrically) occupy two cardcage slots. Incremental power required by the options is minimal; for instance, only 305 mW is needed for the iSBC 300 RAM expansion module.



FUNCTIONAL DESCRIPTION

ISBC 300 32K-Byte MULTIMODULE RAM

The ISBC 300 module contains sixteen 16K-byte dynamic RAM devices, sockets for the Intel® 8202A Dynamic computer. It expands the iSBC 86/12A board's on-board dual port RAM capacity from 32K bytes to 64K bytes. The ISBC 300 module contains sixteen 16K-byte dynamic RAM devices, sockets for the Intel® 8202 Dynamic RAM Controller and memory interface latching. To install the iSBC 300 module, the latches and controller from the iSBC 86/12A board are removed and inserted into the sockets on the iSBC 300 module. The add-on board is then mounted onto the iSBC 86/12A board. Pins extending from the controller's and latches' sockets mate with the devices' sockets underneath (see Figure 1). Additional pins mate to supply power and other signals to complete the electrical interface. The module is then secured at three additional points with nylon hardware to insure the mechanical security of the assembly.

To complete the installation, two socketed PROMs are replaced on the iSBC 86/12A board with those supplied with the iSBC 300 kit. These are the on-board memory and MULTIBUS address decode PROMs which allow the iSBC 86/12A board logic to recognize its expanded on-board memory complement.

ISBC 340 16K-byte MULTIMODULE EPROM

The iSBC 340 module expands the iSBC 86/12A Single Board Computer's on-board EPROM capacity from 16K bytes to 32K bytes. It measures 3.3" by 2.8" and consists of a PC board with six 24-pin special sockets. Two of the sockets have extended pins which mate with two of the EPROM sockets on the iSBC 86/12A board. Two of the EPROMs which would have been inserted on the iSBC 86/12A board are then reinserted in the iSBC 340 module. Additional pins also mate for bringing chip selects for the remaining EPROM devices (see Figure 2). The mechanical interface is similar to that used on the iSBC 300 RAM module and consists of two additional mounting holes and the necessary mounting hardware.

The iSBC 340 module supports Intel® 2732A EPROM. One section of the iSBC 86/12A on-board memory and MULTIBUS address decode PROMs (the same decode PROMs mentioned for the iSBC 300 module) is already preprogrammed to support the iSBC 340 module with Intel® 2732A EPROMs. This section is selected through the EPROM configuration switches on the iSBC 86/12A board. The iSBC 340 board can optionally be configured by the user to support Intel® 2758 or 2761 EPROMs by programming new iSBC 86/12A decode PROMs to support these devices. Necessary documentation and PROM map listings are in the iSBC 86/12A Hardware Reference Manual (order number 9803074-01).

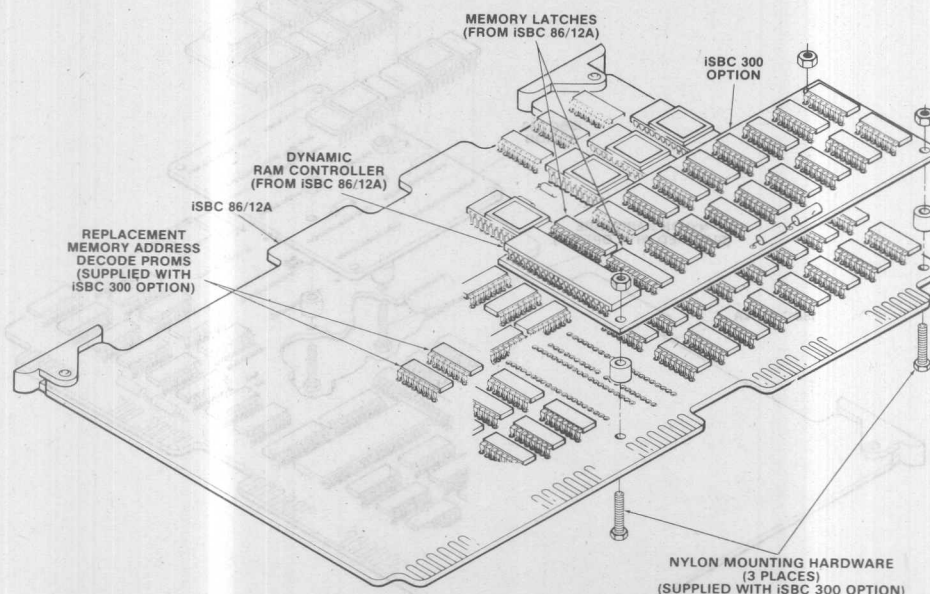


Figure 1. Installation of iSBC 300 MULTIMODULE RAM on iSBC 86/12A Single Board Computer

SPECIFICATIONS

Word Size

8 or 16 bits (16-bit data paths)

Memory Size

iSBC 300 Module — 32,768 bytes of RAM

iSBC 340 Module — 16,384 bytes (max) of EPROM

Access Time

iSBC 300 Module — Read: 1 μ sec, write: 1.2 μ sec

iSBC 340 Module — Standard EPROMs (450 nsec): 1 μ sec, fast EPROMs (350 or 390 nsec): 800 nsec

Interface

The interface for the iSBC 300 and iSBC 340 module options is designed only for Intel's iSBC 86/12A Single Board Computer.

Memory Addressing

On-board RAM

CPU Access

iSBC 86/12A board only (32K bytes) — 00000-07FFFH.

iSBC 86/12A board + iSBC 300 module (64K bytes) — 00000-0FFFFH.

MULTIBUS Access — Jumper selectable for any 8K-byte boundary, but not crossing a 128K-byte boundary.

On-board EPROM

iSBC 86/12A board only (16K-bytes max.) — FF000-FFFFFH (using 2758 EPROMs); FE000-FFFFFH (using 2316E ROMs or 2716 EPROMs); and FC000-FFFFFH (using 2332A ROMs or 2732A EPROMs).

iSBC 86/12A board + iSBC 340 module (32K bytes max) — FE000-FFFFFH (using 2758 EPROMs); FC000-FFFFFH (using 2716 EPROMs); F8000-FFFFFH (using 2732A EPROMs).

On-board EPROM/ROM is not accessible via the MULTIBUS interface.

Auxiliary Power/Memory Protection

The low power memory protection option included on the iSBC 86/12A boards supports the iSBC 300 RAM module.

"Local Only" Memory Protection

The iSBC 86/12A Single Board Computer supports dedication of on-board RAM for on-board CPU access only in 8K, 16K, 24K, or 32K-byte segments. Installation of the iSBC 300 option allows protection of 16K, 32K, 48K, or 64K-byte segments.

Physical Characteristics

	iSBC 300	iSBC 340
Width	5.75"	3.3"
Length	2.35"	2.8"
Height of iSBC 86/12A plus mounted option	.718	.718*
Weight	13 oz.	5 oz.

*Includes EPROMs

All necessary mounting hardware (nylon, screws, spacers, nuts) are supplied with each kit.

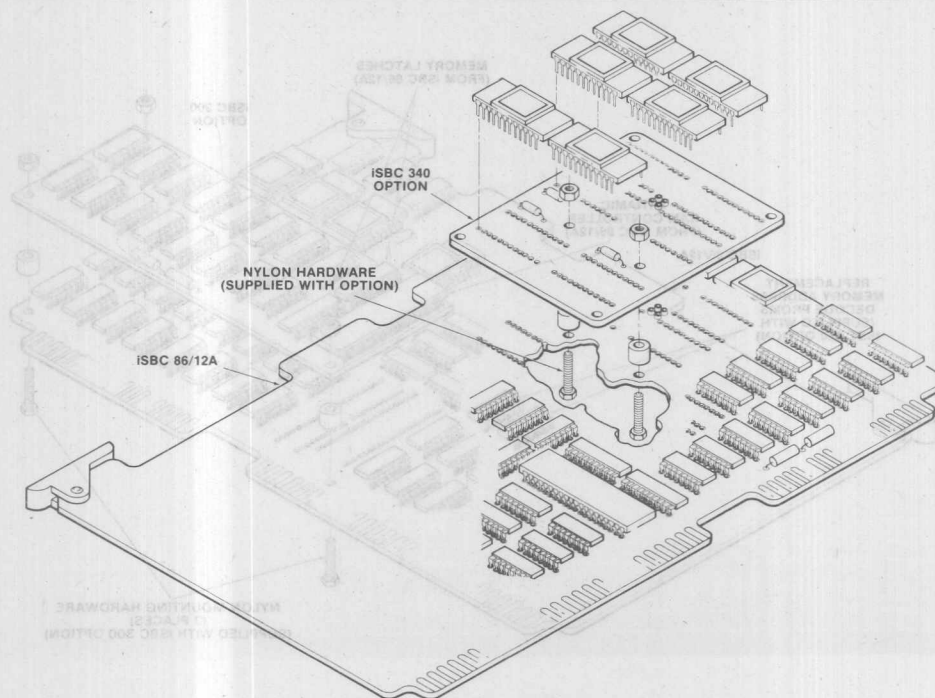


Figure 2. Installation of iSBC 340 MULTIMODULE EPROM Option on iSBC 86/12A Single Board Computer

iSBC 300/340

Electrical Characteristics

DC power requirements:

Voltage	iSBC 300	iSBC 340
+5 \pm 5%	1 mA	120 mA ¹
+12 \pm 5%	24 mA	—
-12 \pm 5%	1 mA	—

Note:

1. Loaded with Intel 2732A EPROMs.

Environmental Characteristics

Operating Temperature — 0° to +55°C

Relative Humidity — to 90% (without condensation)

Reference Manuals

All necessary documentation for the iSBC 300 MULTIMODULE RAM and iSBC 340 MULTIMODULE EPROM/ROM is included in the iSBC 86/12A Hardware Reference Manual; order #9803074-01. (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

ORDERING INFORMATION

Part Number Description

SBC 300 32K byte MULTIMODULE RAM

SBC 340 16K byte MULTIMODULE EPROM

The iSBC 300 and iSBC 340 RAM modules provide simple, low cost expansion of the memory complement available on the iSBC 86/120 and iSBC 86/120 Single Board Computers, respectively. Each module doubles the on-board RAM memory capacity of the host board. The RAM MULTIMODULE options for the host boards offer system designers a new level of flexibility in defining and implementing Intel single board computer systems. Because they expand the memory configuration on-board, they can be accessed as directly as the existing host board memory by eliminating the need for accessing the additional memory via the MULTIBUS system bus.

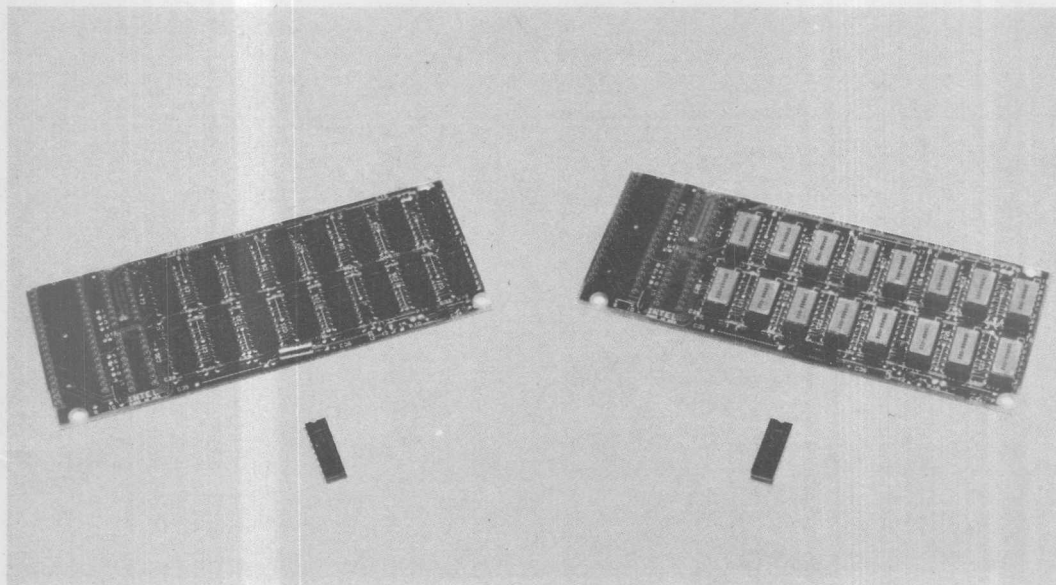


188C 300/340

iSBC® 304 128K BYTE RAM MULTIMODULE™ BOARD iSBC® 300A 32K BYTE RAM MULTIMODULE™ BOARD

- iSBC® 304 module provides 128K bytes of dual port RAM expansion for the iSBC® 86/30 board
- iSBC® 300A module provides 32K bytes of dual port RAM expansion for the iSBC® 86/14 board
- Simple, reliable, mechanical and electrical interconnection
- On-board memory expansion for the iSBC® 86/30 and iSBC® 86/14 Single Board Computers
- On-board memory expansion eliminates MULTIBUS® system bus latency and increases system throughput
- Low power requirements

The iSBC 304 and iSBC 300A RAM modules provide simple, low cost expansion of the memory compliant available on the iSBC 86/30 and iSBC 86/14 Single Board Computers, respectively. Each module doubles the on-board RAM memory capacity of the host board. The RAM MULTIMODULE options for the host boards offer system designers a new level of flexibility in defining and implementing Intel single board computer systems. Because they expand the memory configuration on-board, they can be accessed as quickly as the existing host board memory by eliminating the need for accessing the additional memory via the MULTIBUS system bus.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

January, 1982
Order Number: 210329-001

FUNCTIONAL DESCRIPTION

Each MULTIMODULE contains dynamic RAM devices and sockets for the Intel 8203 dynamic RAM controller and memory interface latching. To install the module, the latches and controller from the host CPU board are removed and inserted into sockets on the RAM MULTIMODULE. The module is then mounted onto the host board. Pins extending from the controller and latch sockets mate with device sockets underneath (see Figure 1). Additional pins mate to supply other signals to complete the electrical interface.

The module is then secured at three additional points with nylon hardware to ensure the mechanical security of the assembly.

To complete the installation, one socketed PROM is replaced on the host CPU board with the one supplied with the MULTIMODULE kit. This is the MULTIBUS address decode PROM which allows the host board logic to recognize its expanded on-board memory compliance.

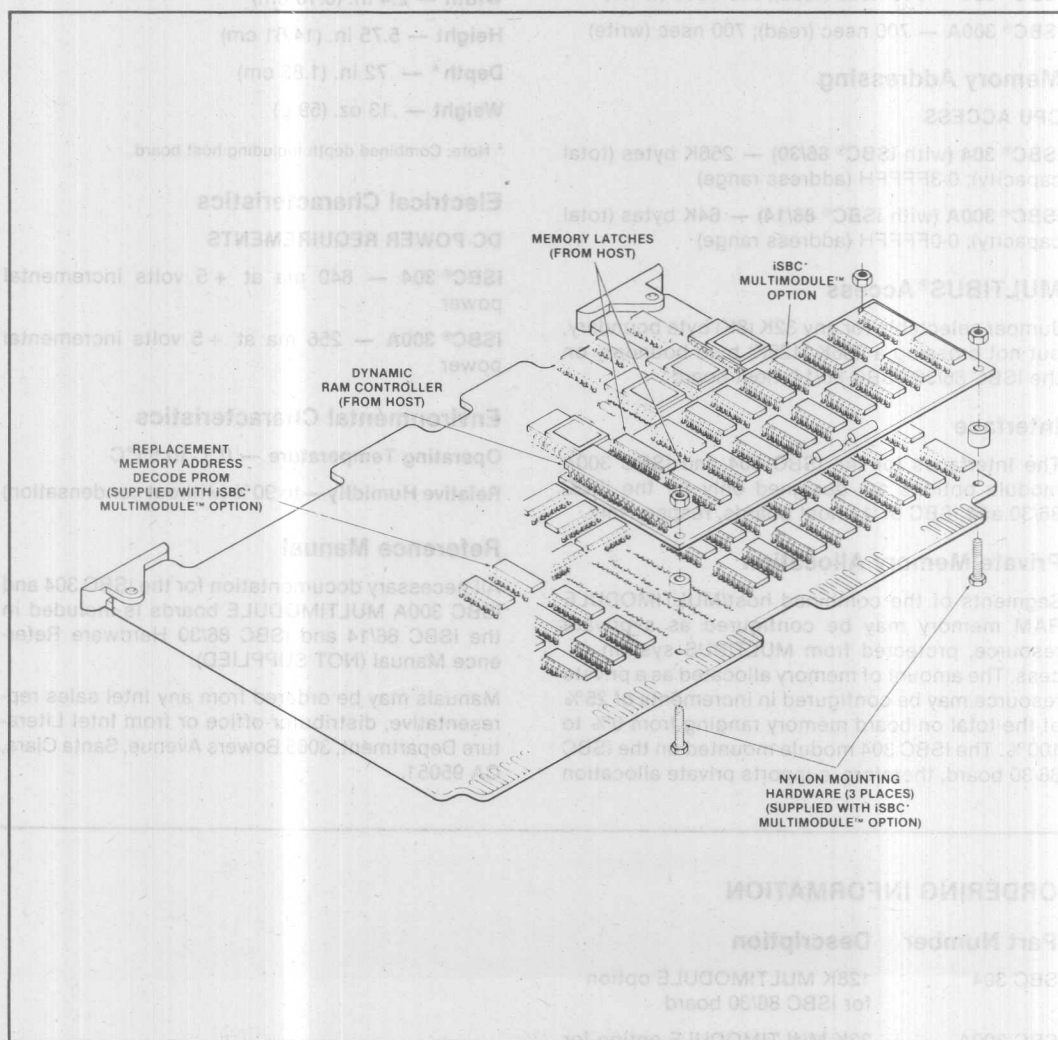


Figure 1. Installation of the MULTIMODULE™ RAM on the Host Single Board Computer

SPECIFICATIONS

Word Size

8 or 16 bits (16-bit data paths)

Memory Size

iSBC® 304 Module — 128K bytes RAM

iSBC® 300A Module — 32K bytes RAM

Cycle Time

iSBC® 304 — 700 nsec (read); 700 nsec (write)

iSBC® 300A — 700 nsec (read); 700 nsec (write)

Memory Addressing

CPU ACCESS

iSBC® 304 (with iSBC® 86/30) — 256K bytes (total capacity); 0-3FFFFH (address range)

iSBC® 300A (with iSBC® 86/14) — 64K bytes (total capacity); 0-0FFFFH (address range)

MULTIBUS® Access

Jumper selectable for any 32K (8K) byte boundary, but not crossing a 256K (128K) byte boundary on the iSBC 86/30 (iSBC 86/14) host board.

Interface

The interfaces for the iSBC 304 and iSBC 300A module options are designed only for the iSBC 86/30 and iSBC 86/14 host boards, respectively.

Private Memory Allocation

Segments of the combined host/MULTIMODULE RAM memory may be configured as a private resource, protected from MULTIBUS system access. The amount of memory allocated as a private resource may be configured in increments of 25% of the total on-board memory ranging from 0% to 100%. The iSBC 304 module mounted on the iSBC 86/30 board, therefore, supports private allocation

of 64K, 128K, 192K, or 256K bytes of RAM memory. The iSBC 300A module mounted on the iSBC 86/14 board supports private allocation of 16K, 32K, 48K, or 64K bytes of RAM memory.

Auxiliary Power

The low power memory protection option included on the CPU host boards supports the RAM modules.

Physical Characteristics

Width — 2.4 in. (6.10 cm)

Height — 5.75 in. (14.61 cm)

Depth* — .72 in. (1.83 cm)

Weight — .13 oz. (59 g)

* Note: Combined depth including host board.

Electrical Characteristics

DC POWER REQUIREMENTS

iSBC® 304 — 640 ma at +5 volts incremental power

iSBC® 300A — 256 ma at +5 volts incremental power

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Relative Humidity — to 90% (without condensation)

Reference Manual

All necessary documentation for the iSBC 304 and iSBC 300A MULTIMODULE boards is included in the iSBC 86/14 and iSBC 86/30 Hardware Reference Manual (NOT SUPPLIED).

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

ORDERING INFORMATION

Part Number Description

SBC 304 128K MULTIMODULE option for iSBC 86/30 board

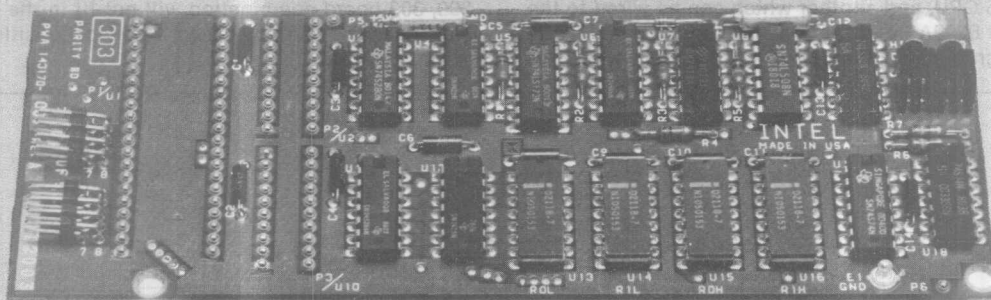
SBC 300A 32K MULTIMODULE option for iSBC 86/14 board

iSBC™ 303 MULTIMODULE™ PARITY

- Add-on parity option for iSBC 86/12A Single Board Computer
- Supports 32KB or 64KB (with iSBC 300 MULTIMODULE RAM) on-board RAM
- Byte parity with programmable odd/even detection/generation

- Two LED error indicators
- Two interrupt requests for error reporting
- No degradation of memory performance
- Memory diagnostic capability

The iSBC 303 MULTIMODULE Parity option provides on-board parity support for the on-board RAM of the iSBC 86/12A Single Board Computer. Memory parity generation/detection is invaluable for those applications in which execution of erroneous instructions or data must be prevented, as in critical process control, medical and financial transaction systems. When used on single board computers in conjunction with MULTIBUS®-based, parity-protected RAM expansion boards in large memory configurations, the iSBC 303 MULTIMODULE Parity option provides consistent protection throughout all system RAM.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE, iCS, iAPX and iMMX. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

FUNCTIONAL DESCRIPTION

The iSBC 303 MULTIMODULE Parity option is a complete subsystem, providing all necessary logic and display to support the iSBC 86/12A on-board RAM. Operation of the iSBC 303 option, once initialized, is transparent to the system, as the option causes no memory performance degradation. If an error is detected and interrupts are enabled, an interrupt request will be issued to the iSBC 86/12A board. Included on-board is the parity generator/checker and parity memory RAM, the interrupt request logic, error display, command register and the necessary interface for address, data and control (inputs) and interrupt requests (outputs) (See Figure 1). The parity generator/checker is a 74S280 MSI device. The memory devices are four Intel® 2118 dynamic RAMs. Parity is generated/detected on a byte basis; for a 32KB RAM configuration, only two of the parity RAMs are used. When the iSBC 303 board is used in conjunction with the iSBC 300 32KB MULTIMODULE RAM option (which provides a total of 64KB on-board RAM), all four parity RAMs are used. Odd or even parity may be programmatically selected through the command register. This feature also provides the ability to "force errors" to verify operation of the board.

Error Reporting

Two interrupt requests are provided: one which can be connected to the NMI input of the Intel 8086 CPU, and another which can be wired to the Intel 8259A interrupt controller and/or Intel 8255A Programmable Peripheral Interface on the iSBC

86/12A board. The 'NMI request' can be enabled/disabled via the command register or optionally with the NMIMASK signal which can be controlled by the 8255A PPI. Additionally, a power fail request signal originating from the system power supply can be "OR'ed" with the parity NMI request, allowing both signals to be issued to the 8086 CPU.

The error display logic contains two LEDs which indicate errors for high and low bytes. High and low byte error signals can also be connected to the 8255A, to read error status under program control. The LEDs and error signals are controlled by the command register.

Base-board Interface

The address, control and data signals are generated by the iSBC 86/12A RAM logic, and are connected to the iSBC 303 module through the sockets of the Intel 8202A RAM controller and the data latches (see Figure 2). The other I/O signals to/from the iSBC 303 board are brought out to an 8-pin connector from which a cable assembly (not supplied) establishes connections to the iSBC 86/12A board. Mechanical integrity is assured by three-point mounting with nylon hardware. The iSBC 86/12A and iSBC 303 boards can be mounted in the end slot of an Intel iSBC 604/614 cardcage. If mounted in other than the top slot, it will physically occupy two slots. The iSBC 86/12A, 300, 303 board combination will not fit in the top slot, and also occupies two cardcage positions when mounted.

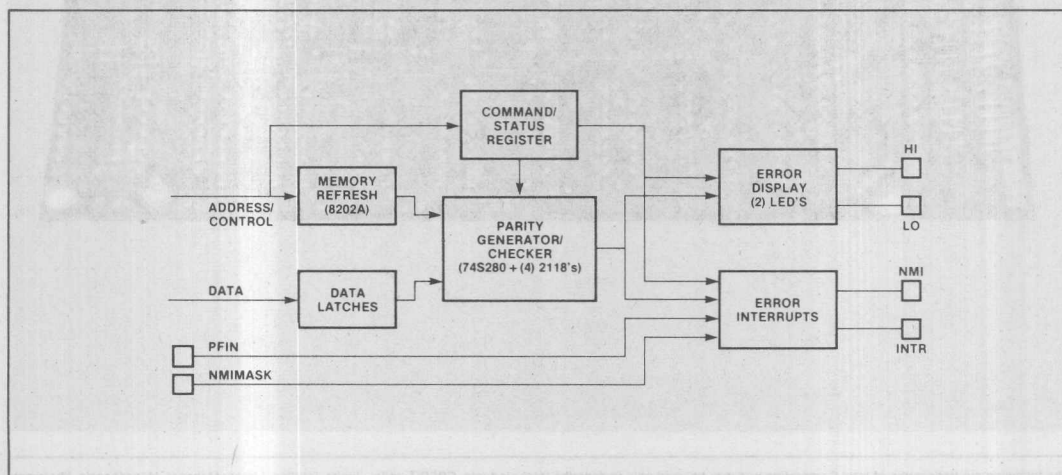


Figure 1. iSBC™ 303 Functional Block Diagram

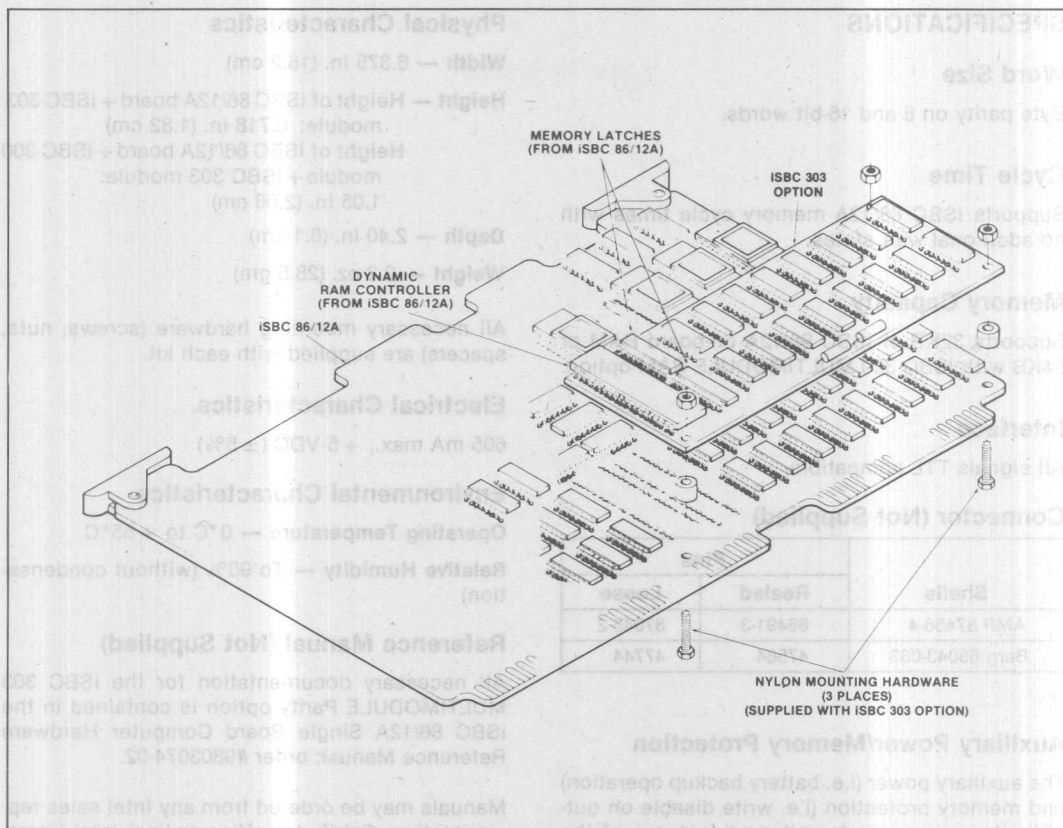


Figure 2. iSBC™ 303 Option Installation

Programmatic Control

The command register is a 4-bit, memory mapped register through which the operation of the iSBC 303 module is controlled (see Figure 3). Read/write access to all bits is available through either location 0H or 400H (jumper selectable). Initialization software should enable the NMI interrupt and the maskable interrupt and LED, as required. It should also select odd or even parity generation (odd is recommended for detection of catastrophic RAM failure).

Diagnostic Capability

Operation of the iSBC 303 module and the integrity of the RAM can be checked with a diagnostic software routine which can 'force errors' in parity memory. This is accomplished by writing data in one parity mode (e.g. odd), toggling the odd/even bit and then reading data with the complementary

parity mode (e.g. even). These reads should cause detectable errors and interrupts will be generated, verifying the error detecting and reporting capability.

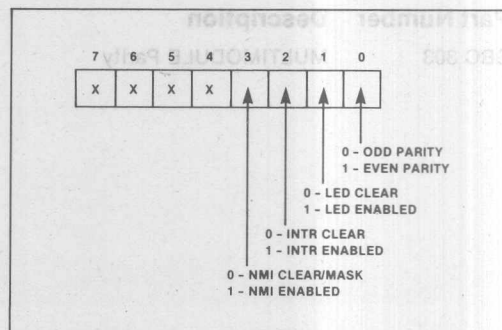


Figure 3. iSBC™ 303 Option Command/Status Register

SPECIFICATIONS

Word Size

Byte parity on 8 and 16-bit words.

Cycle Time

Supports iSBC 86/12A memory cycle times with no additional wait states.

Memory Capacity

Supports 32KB of iSBC 86/12A on-board RAM or 64KB with iSBC 300 MULTIMODULE RAM option.

Interface

All signals TTL compatible.

Connector (Not Supplied)

Shells	Pins	
	Reeled	Loose
AMP 87456-4	86491-3	87045-2
Berg 65043-033	47564	47744

Auxiliary Power/Memory Protection

The auxiliary power (i.e. battery backup operation) and memory protection (i.e. write disable on out-of-limits power supply voltages) features of the iSBC 86/12A Single Board Computer are supported on the iSBC 303 module.

Physical Characteristics

Width — 6.375 in. (16.2 cm)

Height — Height of iSBC 86/12A board + iSBC 303 module: 0.718 in. (1.82 cm)

Height of iSBC 86/12A board + iSBC 300 module + iSBC 303 module:
1.05 in. (2.66 cm)

Depth — 2.40 in. (6.1 cm)

Weight — 2.3 oz. (28.5 gm)

All necessary mounting hardware (screws, nuts, spacers) are supplied with each kit.

Electrical Characteristics

605 mA max., +5 VDC ($\pm 5\%$)

Environmental Characteristics

Operating Temperature — 0°C to +55°C

Relative Humidity — To 90% (without condensation)

Reference Manual (Not Supplied)

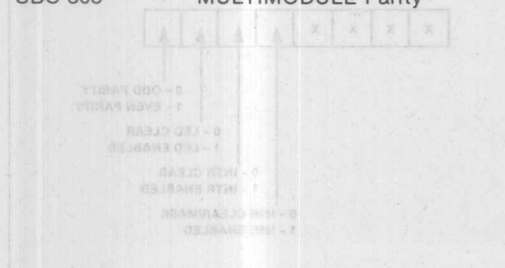
All necessary documentation for the iSBC 303 MULTIMODULE Parity option is contained in the iSBC 86/12A Single Board Computer Hardware Reference Manual; order #9803074-02.

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

ORDERING INFORMATION

Part Number Description

SBC 303 MULTIMODULE Parity



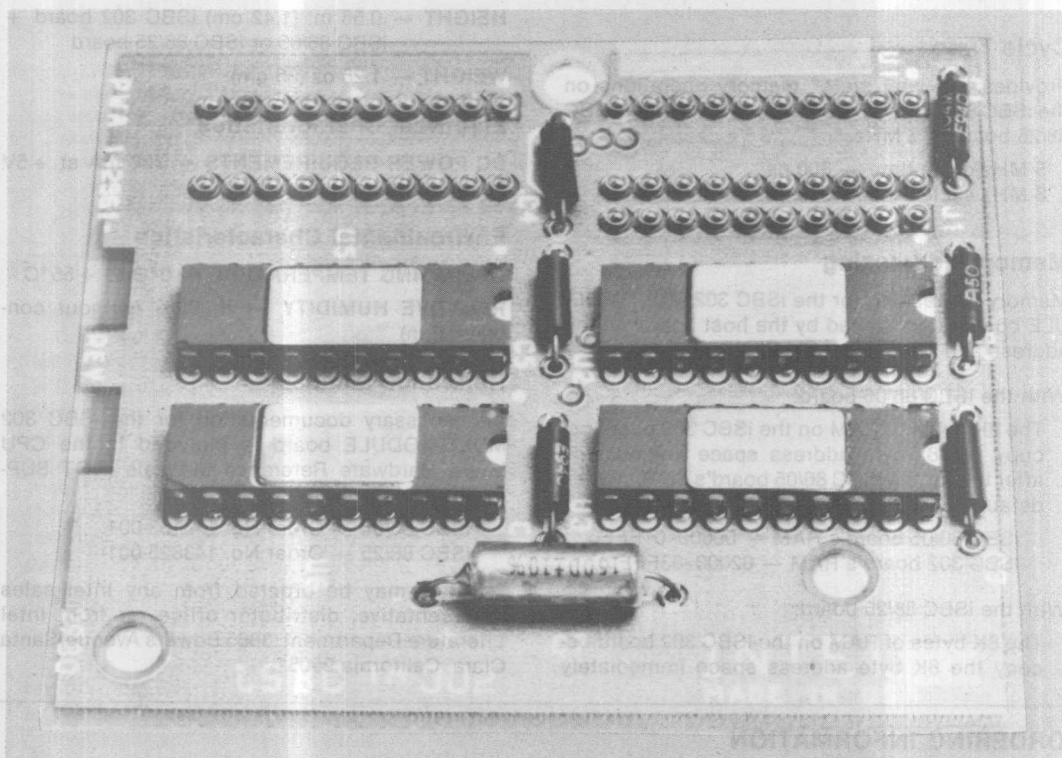
The command register is a 4-bit, memory mapped register through which the operation of the iSBC 303 module is controlled (see Figure 3). Parity write access to all bits is available through either location 0H or 40H (lumper selected). Initialization software should enable the NMI interrupt and the maskable interrupt and LED, as required. It should also select odd or even parity generation (odd is recommended for detection of catastrophic RAM failure).

Operation of the iSBC 303 module and the integrity of the RAM can be checked with a diagnostic software routine which can force errors in parity memory. This is accomplished by writing data in one parity mode (e.g. odd), toggling the odd-even bit and then reading data with the complementary

iSBC™ 302 8K-BYTE MULTIMODULE™ RAM

- Expands on-board memory of the iSBC™ 86/05 and iSBC™ 88/25 Single Board Computers
- Uses four Intel® 2168 static RAMs
- Single +5V supply
- On-board memory expansion eliminates system bus latency and increases system throughput
- Reliable mechanical and electrical interconnection

The Intel iSBC 302 8K-Byte MULTIMODULE RAM provides simple, low-cost expansion to double the RAM capacity on the iSBC 86/05 Single Board Computer to 16K bytes or increase RAM capacity on the iSBC 88/25 Single Board Computer to 12K bytes. This offers system designers a new level of flexibility in implementing system memory. Because the MULTIMODULE memory is configured on-board, it can be accessed as quickly as the standard on-board iSBC 86/05 or iSBC 88/25 memory, eliminating the need for accessing additional memory via the MULTIBUS system bus. As a result, the iSBC 302 board provides a high-speed, cost-effective solution for systems requiring incremental RAM expansion.



FUNCTIONAL DESCRIPTION

The iSBC 302 board measures 2.60" by 2.30" and mounts above the RAM area on the iSBC 86/05 or iSBC 88/25 Single Board Computer. The iSBC 302 MULTIMODULE board contains four 4K x 4 static RAM devices and sockets for two of the RAM devices on the iSBC 86/05 board. With the iSBC 302 module mounted on the iSBC 88/25 board, the two sockets on the iSBC 302 module may be filled with 4K x 4 static RAMs. The two sockets on the iSBC 302 module have extended pins which mate

with two sockets on the base board. Additional pins mate to the power supply and chip select lines to complete the electrical interface. The mechanical integrity of the assembly is assured with nylon hardware securing the module in two places. With the iSBC 86/05 or iSBC 88/25 board mounted in the top slot of an iSBC 604/614 cardcage, sufficient clearance exists for the mounted iSBC 302 option. If the iSBC 86/05 or iSBC 88/25 board is inserted into some other slot, the combination of boards will physically (but not electrically) occupy two cardcage slots.

SPECIFICATIONS

Word Size

8/16 bits

after that of the iSBC 88/25 board's 4K RAM (i.e., default configuration —

iSBC 88/25 board's RAM — 0-0FFF_H

iSBC 302 board's RAM — 01000_H-02FFF_H).

Memory Size

16,384 bytes of RAM

Physical Characteristics

WIDTH — 2.6 in. (6.60 cm)

LENGTH — 2.3 in. (5.84 cm)

HEIGHT — 0.56 in. (1.42 cm) iSBC 302 board +
iSBC 86/05 or iSBC 88/25 board

WEIGHT — 1.25 oz (35 gm)

Cycle Time

Provides "no wait state" memory operations on the iSBC 86/05 board at 5 MHz or 8 MHz or the iSBC 88/25 board at 5 MHz.

5 MHz cycle time — 800 ns

8 MHz cycle time — 500 ns

Electrical Characteristics

DC POWER REQUIREMENTS — 720 mA at +5V incremental power

Memory Addressing

Memory addressing for the iSBC 302 MULTIMODULE board is controlled by the host board via the address and chip select signal lines.

With the iSBC 86/05 board:

The 8K bytes of RAM on the iSBC 302 board occupy the 8K-byte address space immediately after that of the iSBC 86/05 board's 8K RAM (i.e., default configuration —

iSBC 86/05 board's RAM — 00000-01FFF_H

iSBC 302 board's RAM — 02000-03FFF_H).

With the iSBC 88/25 board:

The 8K bytes of RAM on the iSBC 302 board occupy the 8K byte address space immediately

Environmental Characteristics

OPERATING TEMPERATURE — 0°C to +55°C

RELATIVE HUMIDITY — to 90% (without condensation)

Reference Manuals

All necessary documentation for the iSBC 302 MULTIMODULE board is included in the CPU board Hardware Reference Manuals (NOT SUPPLIED).

iSBC 86/05 — Order No. 143153-001

iSBC 88/25 — Order No. 143825-001

Manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

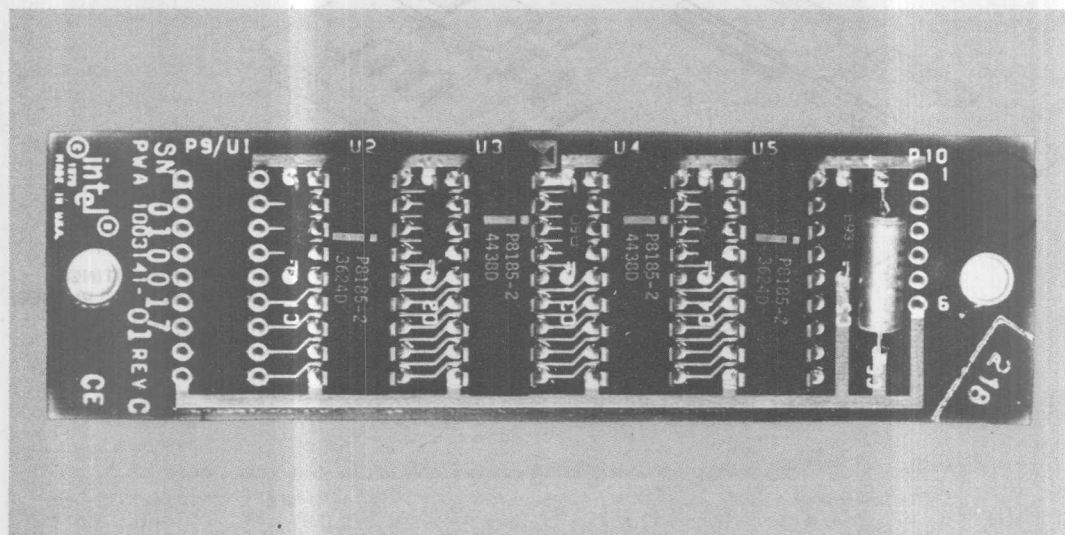
SBC 302 8K-Byte MULTIMODULE RAM



iSBC™ 301 4K-BYTE RAM MULTIMODULE™ BOARD

- On-board memory expansion to 8K bytes for iSBC™ 80/24 and iSBC™ 88/40 Single Board Computers
- Provides 4K bytes of static RAM directly on-board
- Uses 5 MHz (8185-2) RAMs
- Single +5V supply
- 0.5 watts incremental power dissipation
- On-board memory expansion eliminates MULTIBUS system bus latency and increases system throughput
- Reliable mechanical and electrical interconnection

The Intel iSBC 301 4K-Byte RAM MULTIMODULE Board provides simple, low cost expansion to double the RAM capacity on the iSBC 80/24 or iSBC 88/40 Single Board Computer to 8K bytes. This offers system designers a new level of flexibility in defining and implementing system memory requirements. Because memory is configured on-board, it can be accessed as quickly as the existing iSBC 80/24 or iSBC 88/40 memory, eliminating the need for accessing the additional memory via the MULTIBUS system bus. As a result, the iSBC 301 board provides a high speed, cost effective solution for systems requiring incremental RAM expansion. Incremental power required by the iSBC 301 module is minimal, dissipating only 0.5 watts.



FUNCTIONAL DESCRIPTION

The iSBC 301 board measures 3.95" by 1.20" and mounts above the RAM area on the iSBC 80/24 or iSBC 88/40 single board computer. It expands the on-board RAM capacity from 4K bytes to 8K bytes. The iSBC 301 MULTIMODULE board contains four 1K byte static RAM devices and a socket for one of the RAM devices on the iSBC 80/24 or iSBC 88/40 board. To install the iSBC 301 MULTIMODULE board, one of the RAMs is removed from the host board and inserted into the socket on the iSBC 301 board. The add-on board is then mounted into the vacated RAM socket on the host board. Pins ex-

tending from the RAM socket mate with the device's socket underneath (see Figure 1). Additional pins mate to the power supply and chip select lines to complete the electrical interface. The MULTIMODULE board is then secured at two additional points with nylon hardware to insure mechanical security of the assembly. With the iSBC 80/24 or iSBC 88/40 board mounted in the top slot of an iSBC 604 or iSBC 614 cardcage, sufficient clearance exists for mounting the iSBC 301 option. If the iSBC 80/24 or iSBC 88/40 board is inserted into some other slot, the combination of boards will physically (but not electrically) occupy two cardcage slots.

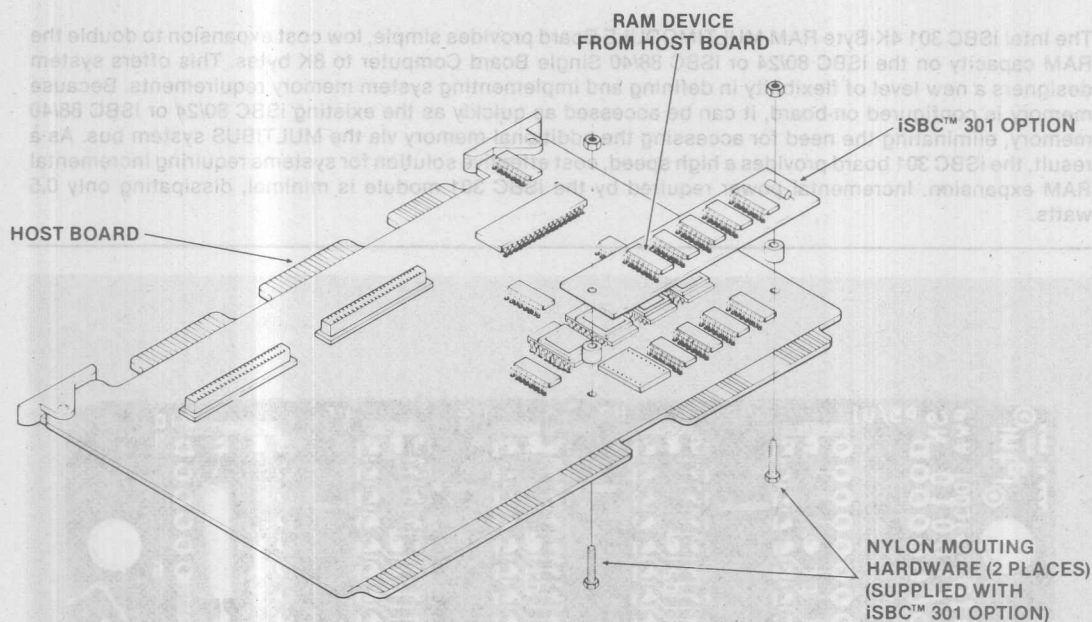


Figure 1. Installation of iSBC™ 301 4K-Byte RAM MULTIMODULE™ Board

SPECIFICATIONS

Word Size

8 bits

Memory Size

4096 bytes of RAM

Access Time

Read: 140 ns (from READ command)

200 ns (from ALE)

Write: 150 ns (from READ command)

190 ns (from ALE)

Memory Addressing

Memory addressing for the iSBC 301 4K-Byte RAM MULTIMODULE Board is controlled by the host board via the address and chip select signal lines and is contiguous with the host board RAM.

iSBC 80/24 and iSBC 301 board: 02000-02FFF

iSBC 88/40 and iSBC 301 board: 00000-01FFF

Physical Characteristics

Width — 1.20 in. (3.05 cm)

Length — 3.95 in. (10.03 cm)

Height — .44 in. (1.12 cm) iSBC 301 Board

.56 in. (1.42 cm)

iSBC 301 Board + host board

Weight — .69 oz. (19 gm)

Electrical Characteristics

DC Power Requirements:

10 mA at +5 Volts incremental power

Environmental Characteristics

Operating Temperature — 0° to +55° C

Relative Humidity — to 90% (without condensation)

Reference Manuals

All necessary documentation for the iSBC 301 MULTIMODULE board is included in the CPU board Hardware Reference Manual (NOT SUPPLIED)

iSBC 80/24 — Order No. 142648-001

iSBC 88/40 — Order No. 124978-001

Manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

SPECIFICATIONS

Part Number Description

SBC 301	4K Byte RAM MULTIMODULE Board
---------	-------------------------------

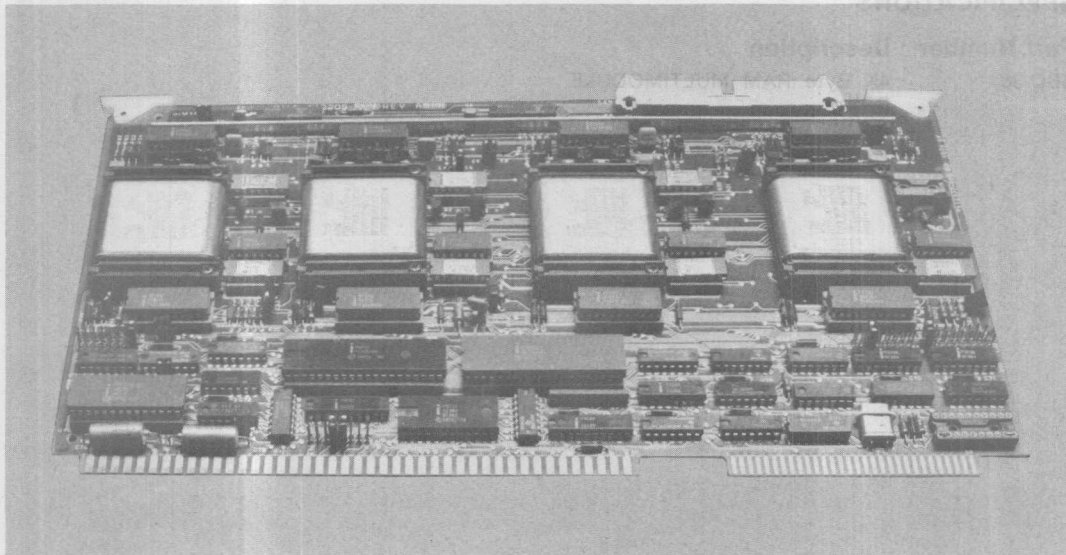
iSBC® 254S BUBBLE MEMORY BOARD

- Capacity up to 512K Bytes of Bubble Memory Storage
- Automatic Error Correction Capability
- Operates from Standard +5V and +12V Power Supplies
- High-Density Storage
- DMA Capability
- Non-Volatile Storage
- High Reliability Even Under Harsh and Rugged Environments
- Average Access Time of 48ms
- Burst Data Rate up to 200K Bytes per Second
- Software Compatible with the iRMX™ Operating System
- Powerfail Data Protection

The iSBC 254S board is a completely assembled and tested non-volatile read/write memory utilizing the Intel 7110 one-megabit bubble memory. This board is offered with one, two, or four 7110 bubble memories, thus yielding capacities of 128K, 256K, or 512K bytes.

Software support is provided under both iRMX/80 and iRMX/86 operating systems, and DMA capability provides the user with considerable flexibility and control. Because of the solid-state nature of this technology, the iSBC 254S board is ideally suited for applications in harsh or rugged environments.

The iSBC 254S board is compatible with 16-bit addressing for 8-bit processors and with 20-bit addressing for 16-bit processors.



The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products: i, Int_l, INTEL, INTELLEC, MCS, Im, iCS, ICE, UPI, BXP, iSBC, iSBX, INSITE, iRMX, CREDIT, RMX/80, μScope, Multibus, PROMPT, Promware, Megachassis, Library Manager, MAIN MULTIMODULE, and the combination of MCS, ICE, SBC, RMX, or iCS and a numerical suffix; e.g., iSBC-80.

© INTEL CORPORATION, 1983.

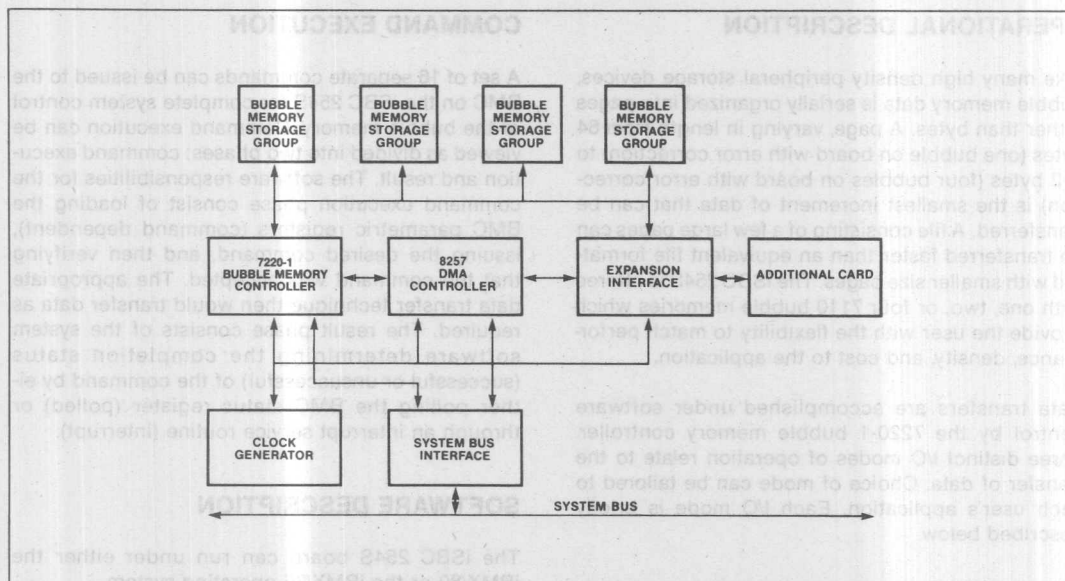


Figure 1. iSBC® 254S Board, Block Diagram

SPECIFICATIONS

Memory Size

128K, 256K, or 512K bytes

Interface

All address, data, and control signals are TTL-compatible and Intel MULTIBUS® system compatible.

Performance

Maximum Data Rate: 200K bytes/second

Average Access Time: 48ms

Power Supply Requirements

Electrical Characteristics

D.C. Power, supplied through MULTIBUS® connector

Voltage	Tolerance	Power Off/Power Fail Decay Rate	Max. Current
+12 Volts	±5%	less than 1.10 volts/msec	1.4A
+5 Volts	±5%	less than 0.45 volts/msec	3.0A

- Voltage sequencing—no restrictions
- Power on voltage rate of rise—no restrictions
- The power supply requirements shown are based on the recommended power fail circuitry.

Connector

86-pin double-sided PC edge connector with 0.40 cm (0.156 in.) contact centers.

Mating Connector: Control Data VFB01E43D0A1 or Viking 2VH43/1ANE5.

Physical Characteristics

Length: 30.48 cm (12 in.)

Height: 17.15 cm (6.75 in.)

Depth: 1.57 cm (0.62 in.)

Note: Because of its depth, the iSBC 254S board requires two card slots in standard MULTIBUS card frame.

Environment

Board Ambient Operating Temperature: 0° to 55°C

Non-Volatile Storage Temperature: -40° to 90°C

Additional Documentation

iSBC 254S Reference Manual (Order No. 113844)

INTEL MULTIBUS Specification (Order No. 9800683)

Memory Components Handbook (Order No. 210830)

OPERATIONAL DESCRIPTION

Like many high density peripheral storage devices, bubble memory data is serially organized into pages rather than bytes. A page, varying in length from 64 bytes (one bubble on board with error correction) to 512 bytes (four bubbles on board with error correction) is the smallest increment of data that can be transferred. A file consisting of a few large pages can be transferred faster than an equivalent file formatted with smaller size pages. The iSBC 254S is offered with one, two, or four 7110 bubble memories which provide the user with the flexibility to match performance, density, and cost to the application.

Data transfers are accomplished under software control by the 7220-1 bubble memory controller. Three distinct I/O modes of operation relate to the transfer of data. Choice of mode can be tailored to each user's application. Each I/O mode is briefly described below.

I/O Modes for Data Transfer

DMA (Direct Memory Access) I/O Mode is the highest performance mode of data transfer. An on-board INTEL 7220-1 Bubble Memory Controller (BMC) and INTEL 8257 DMA controller work in conjunction to generate the MULTIBUS handshake protocol signals. Once a data block transfer begins in this mode, CPU involvement is not required until the entire transfer is completed. This frees the CPU to perform other tasks during DMA transfers. A single data block transfer can be up to 16K bytes in length.

Interrupt I/O Mode requires moderate CPU involvement for data transfers. The BMC generates an on-board interrupt signal when the BMC FIFO (First In-First Out) buffer is either half full (bubble read operation) or half empty (bubble write operation). The interrupt signal is translated to MULTIBUS interrupt line via a jumper. Using this interrupt-driven scheme, software is responsible for performing the appropriate transfer of data (typically 22 bytes) to or from the FIFO buffer when the interrupt occurs.

Polled I/O Mode is the most simple. However, it is also the most demanding of CPU time. System software must determine when to transfer data to or from the FIFO by continually polling a status bit in the BMC status register. The status bit indicates presence or absence of data in the FIFO on a byte-by-byte basis.

COMMAND EXECUTION

A set of 16 separate commands can be issued to the BMC on the iSBC 254S for complete system control of the bubble memory. Command execution can be viewed as divided into two phases: command execution and result. The software responsibilities for the command execution phase consist of loading the BMC parametric registers (command dependent), issuing the desired command, and then verifying that the command was accepted. The appropriate data transfer technique then would transfer data as required. The result phase consists of the system software determining the completion status (successful or unsuccessful) of the command by either polling the BMC status register (polled) or through an interrupt service routine (interrupt).

SOFTWARE DESCRIPTION

The iSBC 254S board can run under either the iRMX/80 or the iRMX/86 operating system.

Under the iRMX/80 operating system a set of two iRMX/80 software tasks perform data transfers. Bubble I/O (BUBIO) provides the interface routines for data storage and retrieval. A second task, the Bubble Manager (BMGR) keeps track of free or available space on the bubble memory device. BMGR operates very similar to the iRMX/80 free space manager by allocating portions of bubble memory space at the request of a user task. BUBIO can be configured to run independent of BMGR.

Under the iRMX/86 operating system, the iSBC 254S board is supported as an integral part of the I/O System Software. The iRMX 86 I/O System Software is implemented as a set of file drivers to support particular types of files and device drivers to provide support to particular devices (i.e., the iSBC 254S Board). Each type of file has its own file driver and each device has its own device driver. This provides great flexibility and device independence since application tasks communicate with file drivers, not with device drivers.

Bubble memory drivers are included in the standard iRMX/86 and iRMX/88 package. Bubble memory drivers for iRMX 80 are available through Insite™ —INTEL's Software Index and Technology Exchange Library.

iSBX™ 251 and 251C BUBBLE MEMORY MULTIMODULE™ BOARD

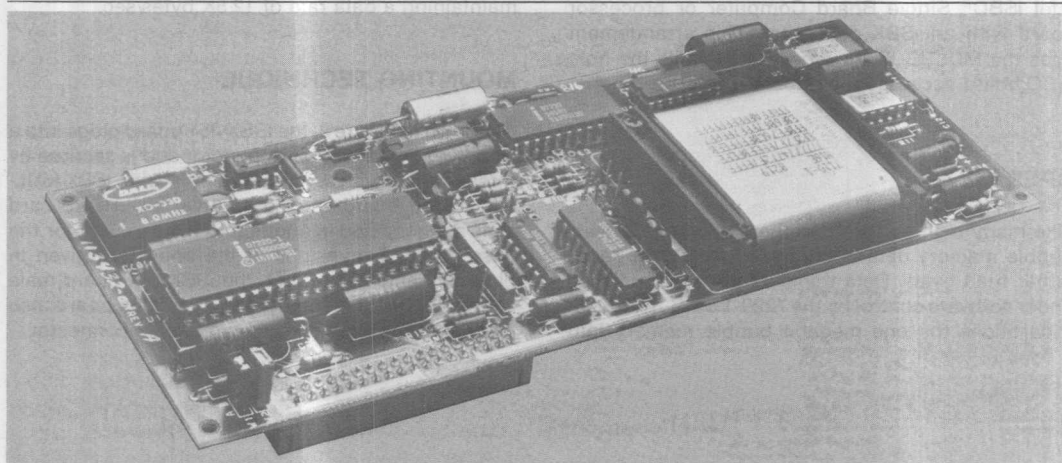
- **iSBX™ 251** 0-60°C
- **iSBX™ 251C** 10-40°C
- **iSBX™ MULTIMODULE™ Bus Compatible**
- **Capacity: 128K Bytes Bubble Memory Storage**
- **Performance:**
 - Average Access Time: 48ms
 - Burst Data Rate: Up to 50K Bytes/Sec
- **Compatibility with Host DMA Controller**

- **Non-Volatile Storage**
- **High Reliability Under Harsh Environments**
- **Fast Access Storage Option on iPDS™ System**
- **Automatic Error Correction**
- **Operates from Standard +5V and +12V Power Supplies**
- **Power Fail Data Protection**
- **Low Power Consumption**

The Intel iSBX 251 and iSBX 251C bubble memory MULTIMODULE boards are completely assembled and tested Non-Volatile 128K-byte memory boards based on the Intel 7110 one-megabit bubble memory and support chips. The iSBX 251 and iSBX 251C boards are Intel's easiest to use bubble solutions. The iSBX 251 and iSBX 251C MULTIMODULE boards may be designed into Intel SBC products with iSBX connectors as well as into any user manufactured microprocessor board. The bubble memory support circuitry and SBX connector on the iSBX 251 and iSBX 251C boards provide the user with a simple interface to the bubble memory.

The iSBX 251 and iSBX 251C boards are featured as an option on the new Intel Personal Development System as a fast access storage option designed to emulate disk. Typically, the bubble memory option provides a 2X improvement in system performance when compared with a floppy disk. Use of the iSBX 251 or iSBX 251C board with the iPDS system enhances iPDS™ system portability, performance and reliability.

The iSBX 251 and iSBX 251C boards differ in specified operating temperature ranges. The iSBX 251 board operating temperature is 0-60°C. The iSBX 251C boards operating temperature is 10-40°C. These boards plug into any Intel iSBX single board computer or other processor board with an iSBX connector. The iSBX 251 board meets Intel iSBX specifications.



The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products: BXP, CREDIT, i, ICE, iCS, iM, Inside, Int'l, INTEL, Inteleview, Intellink, Inteltec, IMMX, iOSP, iPDS, iRMX, iSBC, iSBX, Library Manager, MCS, MULTIMODULE Megachassis, Micromainframe, MULTIBUS, Multichannel, Plug-A-Bubble, PROMPT, Promware, RUP1, RMX/80, System 2000, UPI, and the combination of iCS, iRMX, iSBC, iSBX, ICE, iICE, MCS, or UPI and a numerical suffix. Intel Corporation Assumes No Responsibility for the use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Patent Licenses are implied. © INTEL CORPORATION, 1982.

© INTEL CORPORATION, 1983

FUNCTIONAL DESCRIPTION

The Intel iSBX 251 and iSBX 251C bubble memory MULTIMODULE boards are completely assembled and tested non-volatile memory boards. They consist of the Intel 7110 bubble memory and support circuitry mounted on a double-wide MULTIMODULE board. The bubble memory support circuitry includes the Intel 7220-1 Bubble Memory Controller (BMC) through which the host processor communicates with the bubble storage. See Figure 1 for a system and bubble memory block diagram.

The BMC provides a convenient 8 bit bidirectional bus that requires only two port or I/O addresses. One port is used to transfer data while the other is used to send commands or view operational status. A set of sixteen commands are available to initiate and monitor a bubble memory data transfer. (Refer to the 7220-1 data sheet for more detailed information on the BMC commands).

The iSBX 251 provides the designer with three I/O modes of data transfer for complete flexibility. I/O mode selection is accomplished through the use of on-board jumpers and user software:

1. Polled
2. Interrupt-driven mode
3. Direct Memory Access (DMA) mode

DMA mode requires the use of a DMA controller on the host board.

The iSBX 251 and iSBX 251C boards plug into any Intel iSBC® Single Board Computer or processor board with an iSBX connector. This arrangement frees the MULTIBUS for other traffic while the host iSBC board accesses the bubble memory.

OPERATION

Like many high density peripheral storage devices, bubble memory data is organized serially in pages rather than bytes. Data transfers are accomplished under software control by the 7220-1 BMC. The 7220-1 partitions the one megabit bubble memory into

2048 pages of either 64 or 68 bytes in length. The page length is dependent upon the use of error detection and correction—64 bytes with error correction and 68 bytes without. Data transfers are specified in terms of whole pages. Therefore the minimum amount of data that can be transferred during one read or write command is 64 or 68 bytes. Automatic error correction may be selected by enabling a flag in the 7220-1 BMC.

The iSBX 251 board can be configured to operate in polled mode, interrupt mode, or DMA mode. In the polled mode, the host processor periodically reads the 7220-1 BMC status register to obtain information about completion or termination of commands, error conditions, and the BMC's readiness to transfer data or accept a new command.

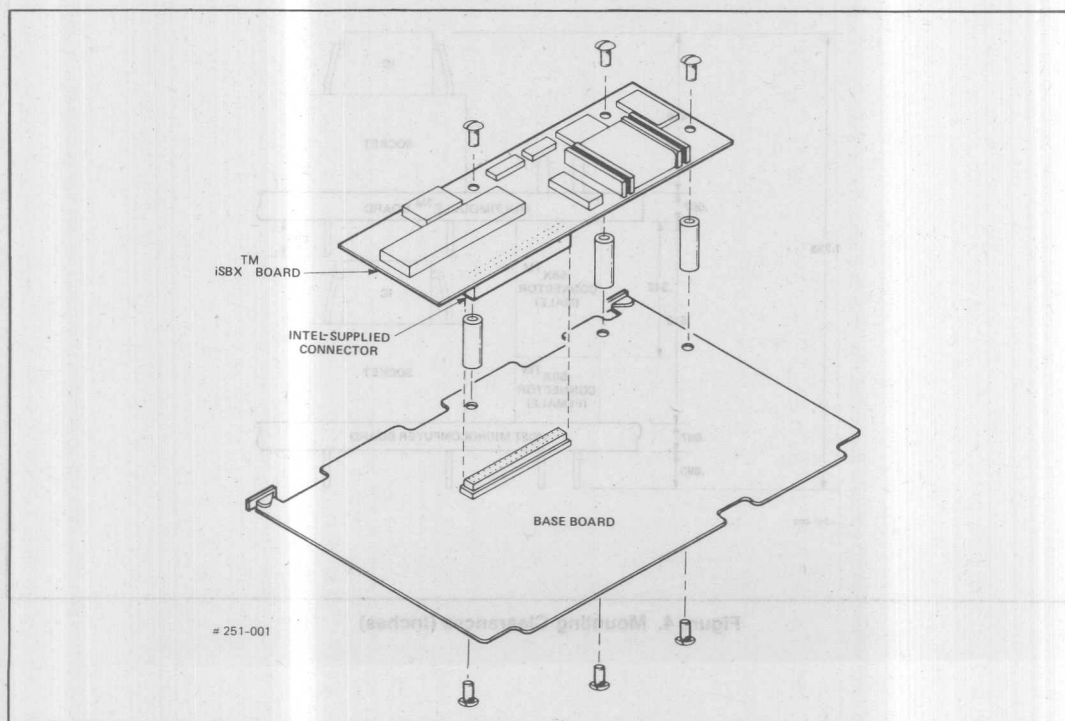
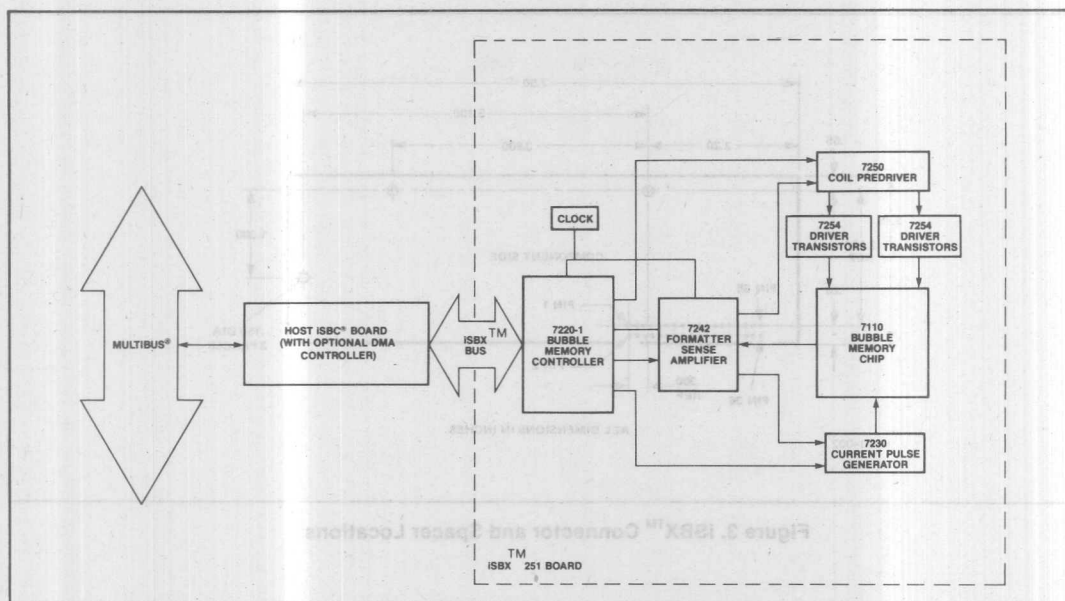
In the interrupt-driven mode, an interrupt is issued by the 7220-1 BMC when its internal buffer is ready to accept 22 bytes of data during a write operation. In a read operation, an interrupt is issued whenever 22 bytes of data are available for reading by the host processor in addition to data transfers. The BMC will also issue an interrupt to indicate the completion of a command or the presence of error conditions.

With the assistance of a direct memory access controller on the board hosting the iSBX 251, the BMC can transfer large blocks of data with a single I/O request. DMA mode makes use of the BMC's hand-shaking ability with a DMA controller.

Regardless of the mode of data transfer, the host processor or DMA controller must be capable of maintaining a data rate of 12.5K bytes/sec.

MOUNTING TECHNIQUE

As shown in Figure 2, the iSBX 251 board plugs into a host board via the iSBX connector and is secured by three spacers with screws. A double-wide iSBX MULTIMODULE board is used and two MULTIBUS card slots are occupied in addition to the card slot for the base board. Dimensions of the board are given in Figures 3 and 4. Although the iSBX 251 board male connector has the standard 36 pins, this board also plugs into the expanded 44-pin female connector.



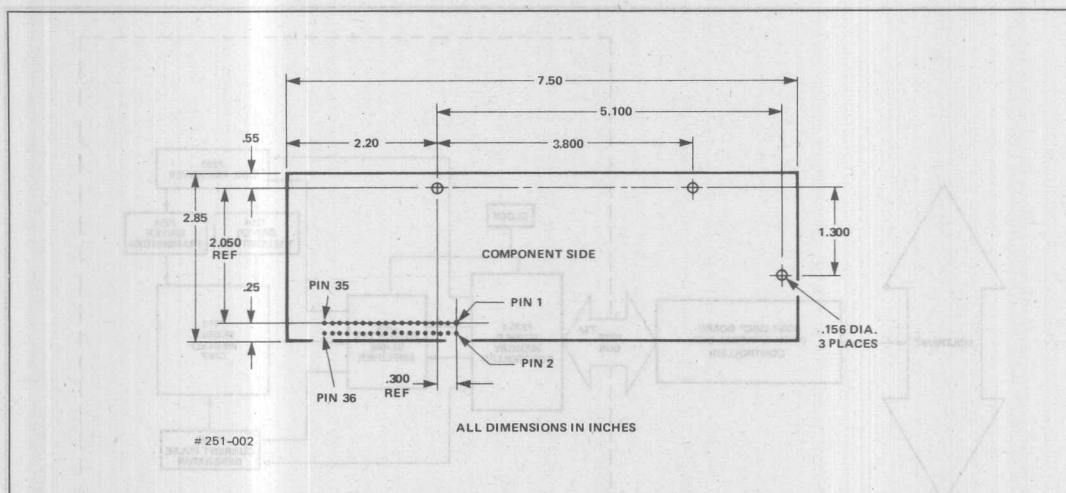


Figure 3. iSBX™ Connector and Spacer Locations

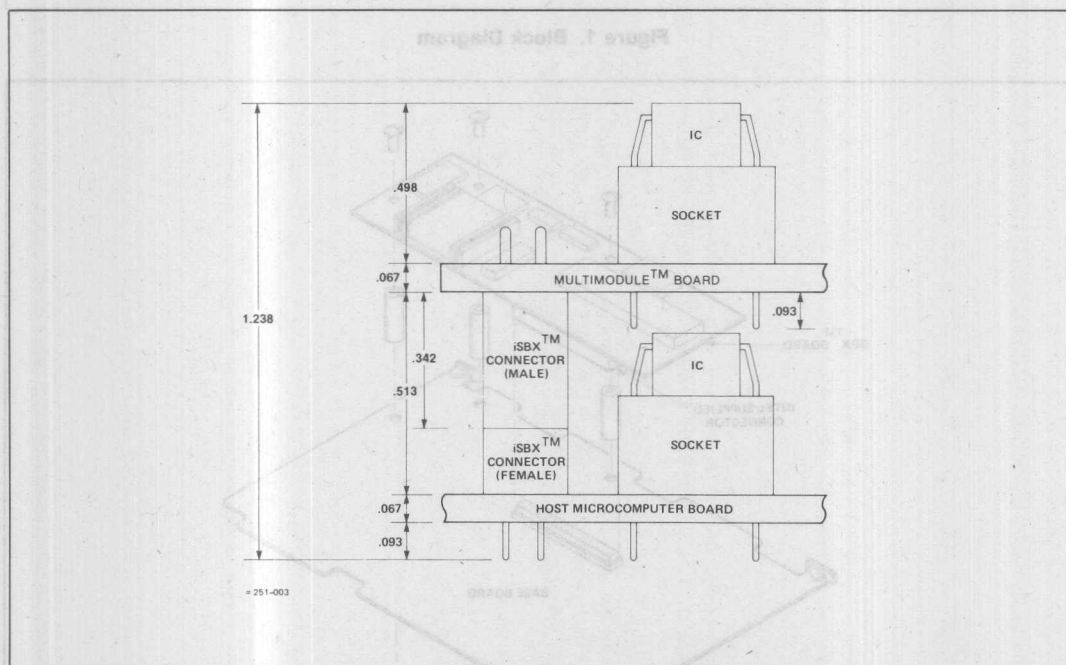


Figure 4. Mounting Clearances (Inches)

SPECIFICATIONS

Storage Capacity

- 128K Eight-Bit Bytes
- 2048 Pages
- Page Length:
 - 64 bytes with ECC
 - 68 bytes without ECC

Physical Characteristics

Width 7.24 cm (2.85 in.)
 Length 19.05 cm (7.50 in.)
 Height 1.27 cm (0.498 in.)
 Weight 362.9 gm (12.8 oz.)

Environment

iSBX 251 board 0-60°C Ambient
 iSBX 251C board 10°-40°C Ambient
 Temperatures quoted are ambient with 100 lfm air flow.

Operational Modes

Polled, Interrupt Driven, or DMA (with Host DMA Controller)

Electrical Requirements

D.C. power, supplied through iSBX connector:

Voltage	D.C. Tolerance	Power Off/Power Fail Decay Rate	Maximum Current
+ 12 Volts	± 5%	less than 1.10 volts/msec	400mA
+ 5 Volts	± 5%	less than 0.45 volts/msec	365mA

Performance

Maximum Data Rate 400K bits/sec
 Average Access Time 48 ms
 Average Transfer Rate 68K bits/sec

Interface Requirements

- TTL compatible
- iSBX 251 male connector plugs into 36-pin or 44-pin host female connector Intel No. 7906.

Relative Humidity:

0% to 95% without condensation

Non-Volatile Storage temperature:

iSBX 251C board - 20 to + 75°C
 iSBX 251 board - 40 to + 90°C

Additional Documentation

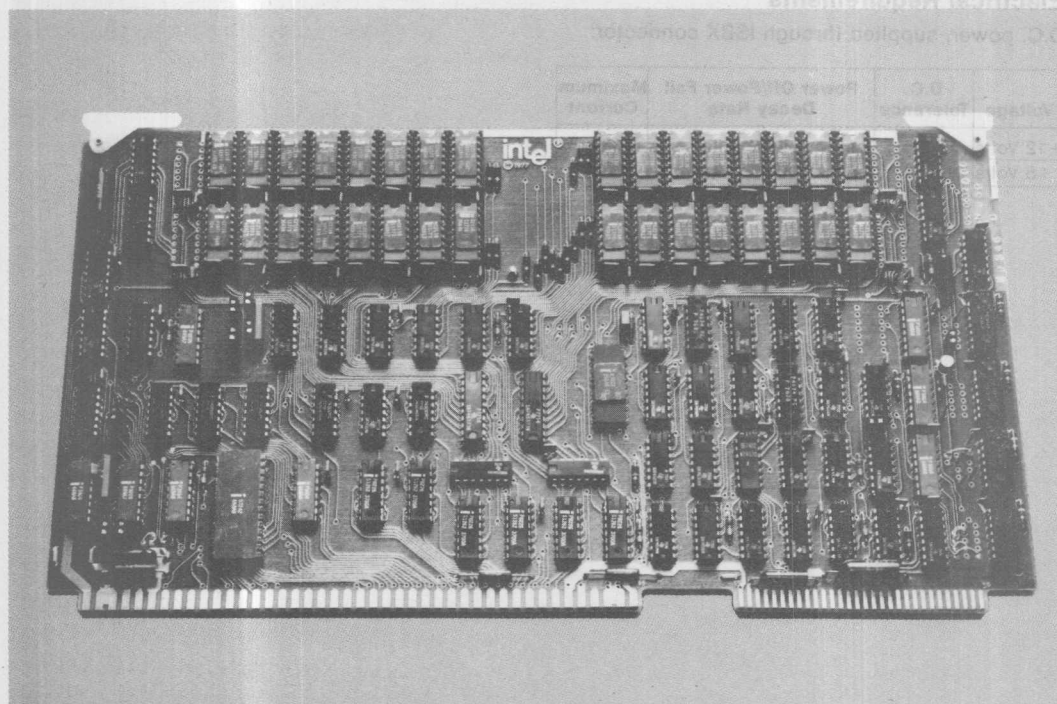
- iSBX251 Technical Manual (Order Number 112924)
- iSBX Bus Specification (Order Number 142686)
- Memory Components Handbook (Order Number 210830)

*MDS is an ordering code and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

iSBC™ 064 RAM MEMORY BOARDS

- iSBC™ 80, iSBC™ 86 and iSBC™ 88 RAM memory expansion through direct MULTIBUS® interface
- 64K bytes of read/write memory
- On-board hardware for refresh of all dynamic memory elements
- Read/write data buffers
- Auxiliary power bus and memory protect control logic provided for battery backup RAM requirements
- Jumper selectable starting address on any 64K boundary in a megabyte address space
- TTL compatible data, address, and command signal interface

The iSBC 064 RAM Memory Board is a member of Intel's complete line of iSBC memory and I/O expansion boards. Each board interfaces directly to any Intel iSBC 80, iSBC 86 or iSBC 88 single board computer via the MULTIBUS interface to expand RAM memory capacity. The iSBC 064 contains 64K bytes of read/write memory implemented using dynamic RAM memory components. On-board refresh hardware refreshes a portion of RAM memory every 14 microseconds. Each refresh cycle utilizes memory for 585 nanoseconds. If a read or write cycle is in progress when a refresh cycle is scheduled to begin, the refresh cycle is postponed until the end of the cycle. Read/write buffers reside on each board to buffer all data written into or read from the memory array. All data, address, and command signals on the bus interface are TTL compatible.



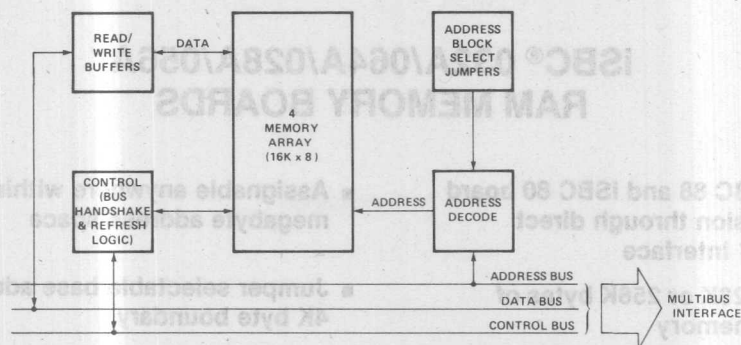


Figure 1. RAM Memory Expansion Boards Block Diagram

SPECIFICATIONS

Word Size

8 bits and 16 bits

Memory Size

65,536 bytes

Access Time

450 ns max

Cycle Times

Read Cycle — 700 ns max

Write Cycle — 600/1240 ns max

Refresh Cycle — 700 ns max

Interface

All address, data, and command signals TTL compatible.

Address Selection

Jumper selection of a 64K byte page in a megabyte address space.

Connectors

Edge Connectors — 86-pin double-sided PC edge connector with 0.156-in. contact centers.

Mating Connector — Viking 3KH43/9AMK12

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM for systems requiring battery back up of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power-down sequences.

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.76 in. (17.15 cm)

Depth — 0.50 in. (1.27 cm)

Weight — 14 oz (415.2 gm)

Electrical Characteristics

DC Power Requirements

Voltage	Normal System Operation (max) ¹	AUX Power RAM Access (max) ²	AUX Power No RAM Access (max)
V _{CC} = +5V ± 5%	I _{CC} = 3.2A	1.7A	1.7A
V _{DD} = +12V ± 5%	I _{DD} = 600 mA	600 mA	120 mA
V _{BB} = -5V ± 5%	I _{BB} = 10 mA	10 mA	3 mA

Notes

1. All current values include AUX power.
2. RAM chips and RAM control logic powered via auxiliary power bus.
3. Power necessary to refresh RAMs and maintain data, as after system power failure.

Environmental Characteristics

Operating Temperature — 0°C to +55°C

Reference Manual

9800488B — iSBC 032/048/064 Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

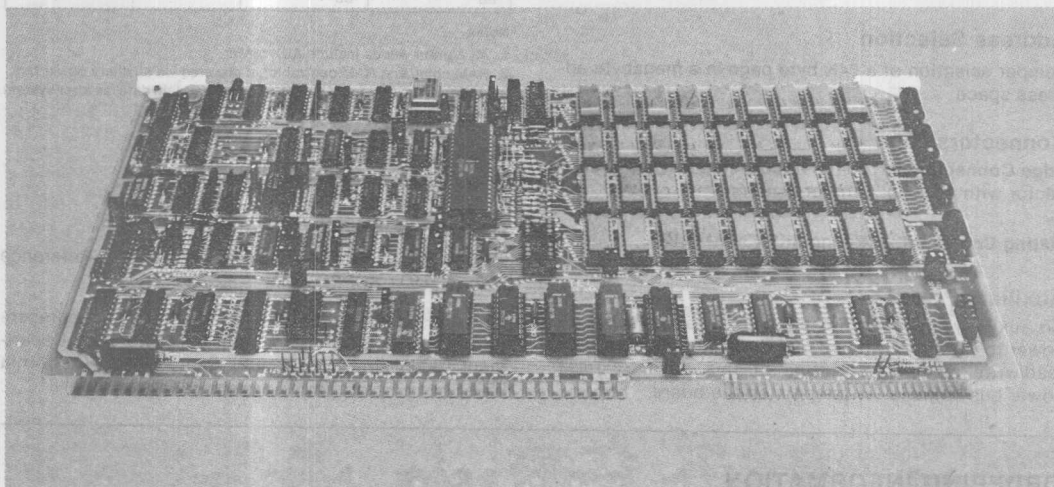
SBC 064 64K-Byte RAM Board

iSBC 032A/064A/028A/056A RAM MEMORY BOARDS

- iSBC 86, iSBC 88 and iSBC 80 board RAM expansion through direct MULTIBUS® interface
- Assignable anywhere within a 16 megabyte address space
- 32K, 64K, 128K or 256K bytes of read/write memory
- Jumper selectable base address on any 4K byte boundary
- On-board parity generator/checker and error status register
- Auxiliary power bus and memory protect control logic for battery backup RAM requirements
- Requires a single +5 volt power supply

The iSBC 032A, iSBC 064A, iSBC 028A and iSBC 056A RAM memory boards are members of Intel's complete line of iSBC memory and I/O expansion boards. Each board interfaces directly to any iSBC 80, iSBC 88 or iSBC 86 Single Board Computer via the MULTIBUS interface to expand system RAM capacity. The iSBC 032A, iSBC 064A, iSBC 028A and iSBC 056A boards contain 32K, 64K, 128K, or 256K bytes of read/write memory implemented using dynamic RAM components. An on-board LSI dynamic RAM controller refreshes a portion of these components every 14 microseconds. Each refresh cycle utilizes memory for 480 nanoseconds (maximum).

The iSBC 032A, iSBC 064A, iSBC 028A and iSBC 056A boards generate byte oriented parity during all write operations and perform parity checking during all read operations. When a parity error is detected, these boards can generate an interrupt on the MULTIBUS interface. In addition, the row and bank of the RAM array containing the error are stored in a Parity Flag Register (see Figure 1). This register is accessible as a MULTIBUS I/O port. An on-board LED also provides a visual indication that a parity error has occurred. To facilitate testing of these boards, parity generation and checking can be changed from even to odd under software control.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE, ICS, IAPX and iMMX. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

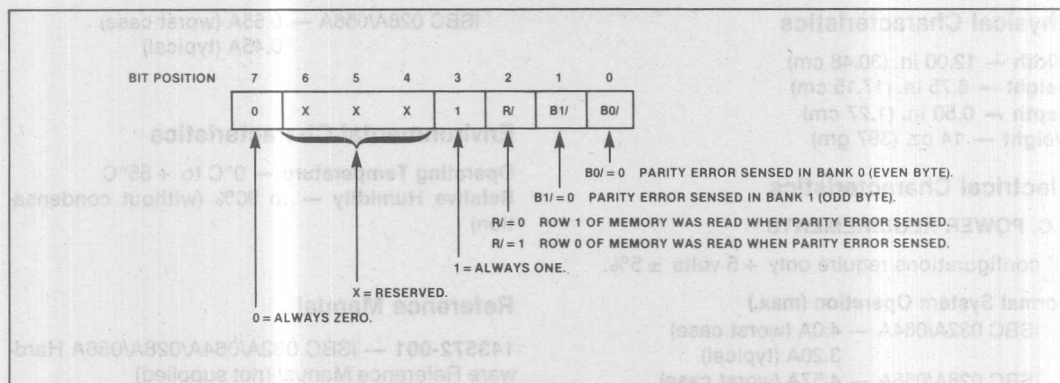


Figure 1. Parity Flag Register Format

SPECIFICATIONS

Word Size

8 bits and 16 bits

Memory Size

32,768 bytes (iSBC 032A); 65,536 bytes (iSBC 064A); 131,072 bytes (iSBC 028A); or 262,144 bytes (iSBC 056A)

Access Time

iSBC 032A/064A

400 ns max. (worst case)
360 ns max. (typical)

iSBC 028A

500 ns max. (worst case)
460 ns max. (typical)

iSBC 056A

570 ns max. (worst case)
530 ns max. (typical)

Cycle Times (Worst Case)

Read

iSBC 032A/064A/028A — 600 ns max.
iSBC 056A — 650 ns max.

Write

iSBC 032A/064A/028A — 600 ns max.
iSBC 056A — 650 ns max.

Refresh

iSBC 032A/064A/028A — 480 ns max.
iSBC 056A — 600 ns max.

Interface

All address, data and command signals are TTL compatible.

Address Selection

Memory — Base address is jumper selectable on any 4K byte boundary in a 16 megabyte address space. On-board RAM cannot cross a megabyte address boundary.

Parity Flag Register — The I/O address of the Parity Flag Register is jumper selectable to be between 00H to 0FH or 40H to 4FH.

Connector

Edge connector — 86 pin double-sided PC edge connector with 0.156 in. contact centers.

Mating connector — Viking 3KH43/9AMK12 or equivalent.

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM array for systems requiring battery backup of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM on the board. This input is provided for the protection of RAM contents during system power-down sequences.

Physical Characteristics

Width — 12.00 in. (30.48 cm)
Height — 6.75 in. (17.15 cm)
Depth — 0.50 in. (1.27 cm)
Weight — 14 oz. (397 gm)

Electrical Characteristics

D.C. POWER REQUIREMENTS

All configurations require only +5 volts \pm 5%.

Normal System Operation (max.)

iSBC 032A/064A — 4.0A (worst case)
 3.20A (typical)
 iSBC 028A/056A — 4.57A (worst case)
 3.66A (typical)

Auxiliary Power No RAM Access (max.)

iSBC 032A/064A — 0.41A (worst case)
 0.33A (typical)

iSBC 028A/056A — 0.55A (worst case)
 0.45A (typical)

Environmental Characteristics

Operating Temperature — 0°C to +55°C

Relative Humidity — to 90% (without condensation)

Reference Manual

143572-001 — iSBC 032A/064A/028A/056A Hardware Reference Manual (not supplied)

Manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Dept., 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

SBC 032A 32K-Byte RAM Board with Parity
 SBC 064A 64K-Byte RAM Board with Parity
 SBC 028A 128K-Byte RAM Board with Parity
 SBC 056A 256K-Byte RAM Board with Parity

iSBC® 028CX, 056CX, AND 012CX iLBX™ RAM BOARDS

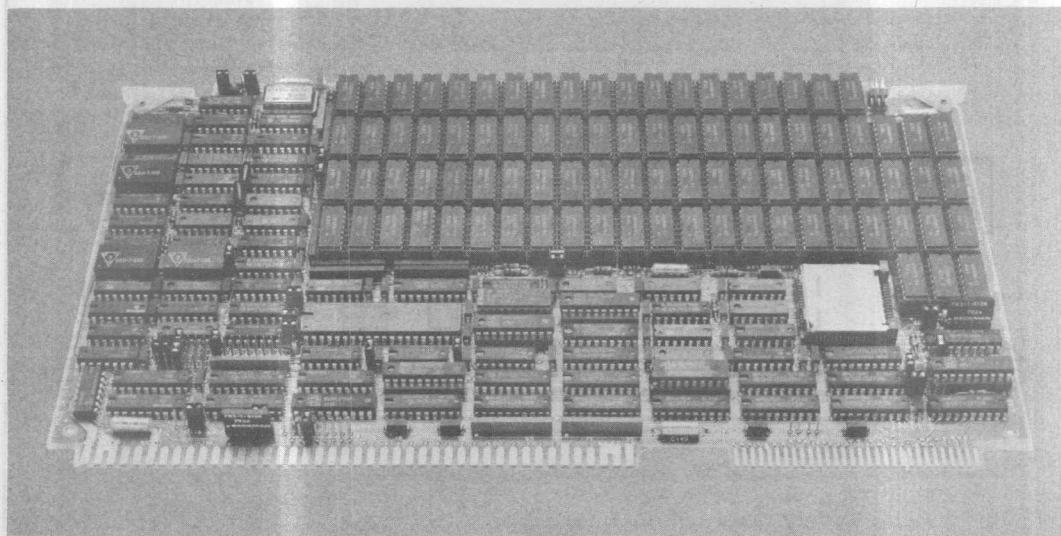
- Dual port capability via MULTIBUS® and iLBX™ Bus Interfaces
- Single bit error correction and double bit error detection via Intel 8206 ECC device
- 128K byte, 256K byte and 512K byte versions available
- Control status register supports multiple ECC operating modes
- Error status register provides error logging by host CPU board
- 16 megabyte addressing capability with base address selectable on 16K byte boundaries
- Supports 8- or 16-bit data transfer and 24-bit addressing
- Auxiliary power bus and memory protect logic for battery back-up RAM requirements

The iSBC® 028CX, iSBC 056CX, and iSBC 012CX RAM memory boards are members of Intel's complete line of iSBC memory and I/O expansion boards. Each board interfaces directly to any iSBC 80, iSBC 88, iSBC 86, iSBC 186 and iSBC 286 Single Board Computers. The dual port feature of the CX series of RAM-boards allows access to the memory of both the MULTIBUS® and the iLBX™ interfaces.

In addition to the dual port features the "CX" series of RAM-boards provide Error Correction Circuitry (ECC) which can detect and correct single bit errors and detect, but not correct, double and most multiple bit errors.

The iSBC 028CX, iSBC 056CX and iSBC 012CX boards contain 128K, 256K or 512K bytes of read/write memory using 64K dynamic RAM components.

Due to the iLBX dual port capability and on-board ECC features of the board they are ideally suited in applications where memory performance and integrity of the stored data is critical, such as financial transactions, process control and medical equipment applications.



FUNCTIONAL DESCRIPTION

General

The iSBC 028CX, 056CX and 012CX RAM boards are physically and electrically compatible with the MULTIBUS interface standard, IEEE P796, as outlined in the Intel MULTIBUS specification. In addition the CX series of RAM-boards are physically and electrically compatible with the iLBX bus interface as outlined in the Intel iLBX Specification.

Dual Port Capabilities

The "CX" series of RAM-boards can be accessed by either the MULTIBUS interface or the iLBX interface. Intel's new Local Bus Extension interface (iLBX bus) is an unarbitrated bus architecture which allows direct transfer of data transfer between the CPU and the memory boards without accessing the MULTIBUS. Due to the unarbitrated nature of the iLBX interface significant improvements in memory access times result, typically 350% to 400%, over MULTIBUS memory access times.

System Memory Size

Maximum system memory size with this series of boards is 16 megabytes. Memory partitioning is independent for the MULTIBUS interface and the iLBX interface.

For MULTIBUS operations, on-board jumpers assign the board to one of four 4-megabyte pages. Each page

is partitioned into 256 blocks of 16K bytes each. The smallest partition on any board in this series is 16K bytes. Jumpers assign the base address (lowest 16K block) within the selected 4-megabyte page.

The iLBX bus memory partitioning differs from the MULTIBUS partitioning in that the iLBX bus address space consists of 256 contiguous blocks of 64K bytes totaling 16 megabytes. As with the MULTIBUS partitioning the base addresses are set with on-board jumpers.

Error Checking and Correcting (ECC)

Error checking and correction is accomplished with the Intel 8206 Error Checking and Correcting device. This ECC component in conjunction with the ECC check bit RAM array provides error detection and correction of single bit errors and detection only of double bit and most multiple bit errors. The ECC circuitry can be programmed via the Control Status Register (CSR) to various modes while error logging is supported by the Error Status Register (ESR). Both CSR and ESR communicate with the master CPU board through a single I/O port.

ECC I/O ADDRESS SELECTION

The processor board communicates with the ECC circuitry via a single I/O port. This port is used for the Control Status Register (CSR) and the Error Status Register (ESR). The CSR is programmed by the user to determine the mode of operation while the ESR provides information about memory errors. The iSBC 028CX, iSBC

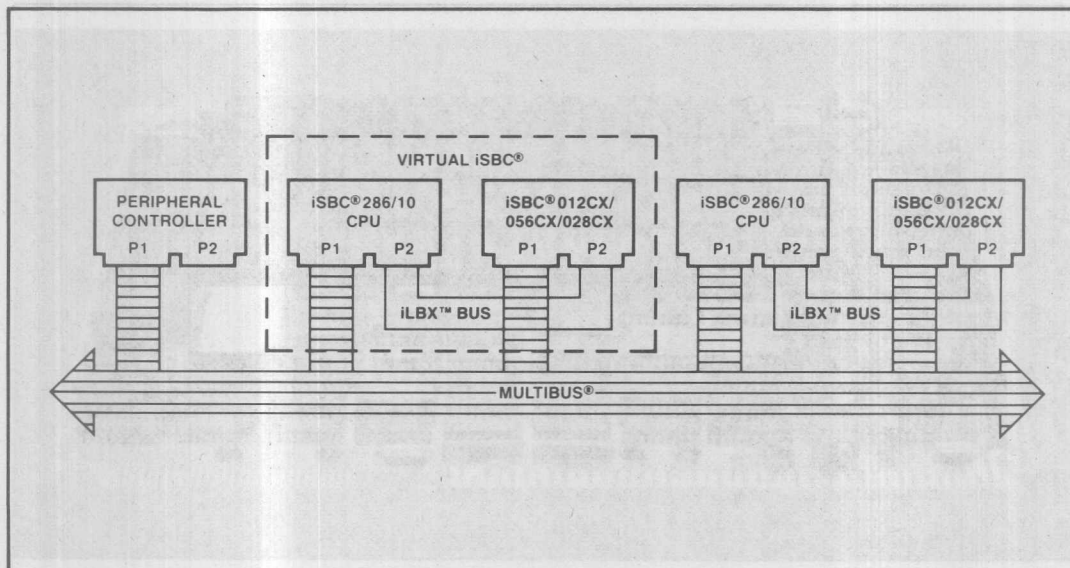


Figure 1. Typical iLBX™ System Configuration

256CX and iSBC 012CX RAM boards are shipped with a Programmable Array Logic (PAL) device which allows selecting one of 9 possible addresses for the I/O port. The actual selection is done by jumper configuration. Additional unprogrammed locations are left in the PAL to allow application specific I/O addresses to be defined.

CONTROL STATUS REGISTER

There are six ECC modes of operation in the "C" Series family of RAM boards. Each mode is obtained by software programming of the CSR from the master iSBC board. The six modes are:

- Interrupt on any error mode
- Interrupt on non-correctable error only mode
- Correcting mode
- Non-correcting mode
- Diagnostic mode
- Examine syndrome word mode

Modes (a) and (b) can be used in conjunction with (c) and (d). The six modes are described below.

Interrupt on Any Error Mode — In this mode the RAM board will interrupt the iSBC processor board when any error (single bit or multiple bit) is detected by the ECC circuitry.

Interrupt on Non-Correctable Error Mode — In this mode the RAM board will interrupt the iSBC processor board only when a non-correctable (multiple bit) error is detected by the ECC circuitry. A multiple bit error is not correctable by the ECC circuitry.

Correcting Mode — In this mode the RAM board corrects any correctable error (single bit error). Errors which are not correctable are not modified. Interrupts are generated depending on the interrupt mode selected.

Non-Correcting Mode — In this mode the RAM board does not correct any error. The ECC circuitry continues to check for errors, but no corrective action is taken. Interrupts continue as described previously.

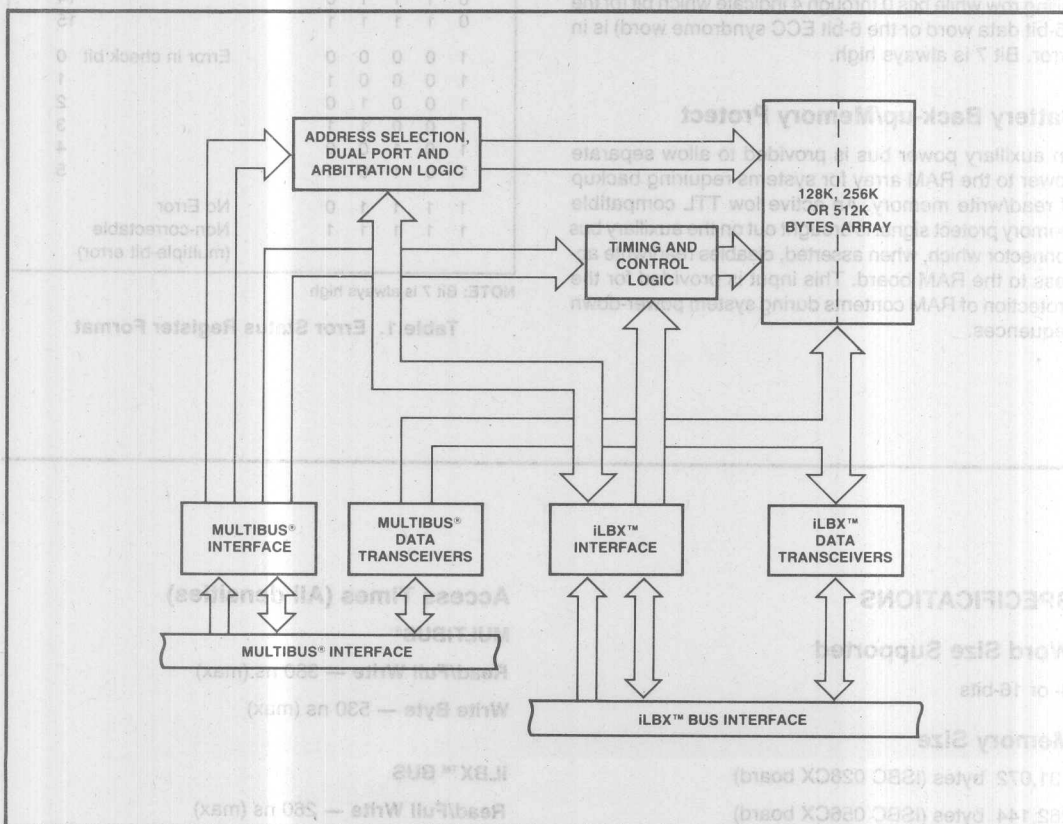


Figure 2. iSBC® 028CX/056CX/012CX Block Diagram

Diagnostic Mode — This mode is used for testing the on-board ECC circuitry. In this mode the write enable strobe to the ECC RAM array is continuously disabled. The diagnostic mode can be used to simulate errors and in conjunction with the "Examine Syndrome Word Mode" examine the check bits generated by the ECC circuitry.

Examine Syndrome Word Mode — This mode, in conjunction with the diagnostic mode, is used for testing the ECC memory. In this mode, the syndrome bits/check bits are clocked into the ESR on every memory read/write cycle, respectively. The ESR translation PROM switches to a transparent mode in the examine syndrome word mode. This allows the actual syndrome word generated by the 8206 ECC device to be examined.

ERROR STATUS REGISTER

This 8-bit register contains information about memory errors. The ESR reflects the latest error occurrence. Table 1 shows the status register format. Bits 5 and 6 show the failing row while bits 0 through 4 indicate which bit (of the 16-bit data word or the 6-bit ECC syndrome word) is in error. Bit 7 is always high.

Battery Back-up/Memory Protect

An auxiliary power bus is provided to allow separate power to the RAM array for systems requiring backup of read/write memory. An active low TTL compatible memory protect signal is brought out on the auxiliary bus connector which, when asserted, disables read/write access to the RAM board. This input is provided for the protection of RAM contents during system power-down sequences.

Bit		Meaning
6	5	
0	0	Error in row 0
0	1	1
1	0	2
1	1	3

Bit					Meaning
4	3	2	1	0	
0	0	0	0	0	Error in data bit 0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	Error in check bit 0
1	0	0	0	1	1
1	0	0	1	0	2
1	0	0	1	1	3
1	0	1	0	0	4
1	0	1	0	1	5
1	1	1	1	0	No Error
1	1	1	1	1	Non-correctable (multiple-bit error)

NOTE: Bit 7 is always high

Table 1. Error Status Register Format

SPECIFICATIONS

Word Size Supported

8- or 16-bits

Memory Size

131,072 bytes (iSBC 028CX board)

262,144 bytes (iSBC 056CX board)

524,288 bytes (iSBC 012CX board)

Access Times (All densities)

MULTIBUS®

Read/Full Write — 380 ns (max)

Write Byte — 530 ns (max)

iLBX™ BUS

Read/Full Write — 260 ns (max)

Write Byte — 440 ns (max)

**Cycle Times (All densities)****MULTIBUS®****Read/Full Write** — 490 ns (max)**Write Byte** — 885 ns (max)**iLBX™ BUS****Read/Full Write** — 375 ns**Write Byte** — 740 ns**NOTE:** If an error is detected, read access time and cycle times are extended to 255 ns (max)**Refresh Cycle Time** — 15.6 μ s**Refresh Delay Time** — 760 ns**Memory Partitioning**

Maximum System memory size is 16M Bytes for both MULTIBUS and iLBX BUS. MULTIBUS partitioning is by Page, Block and Base, while the iLBX BUS is by Block and Base only.

PAGE ADDRESS**MULTIBUS®** — 0-4 megabytes; 4-8 megabytes; 8-12 megabytes; 12-16 megabytes**iLBX™ BUS** — N/A**BLOCK ADDRESS**

Board	MULTIBUS® (16K Bytes)	iLBX™ Bus (64K Bytes)
iSBC® 028C	8 contiguous 16K byte blocks	2 blocks of 64K bytes
iSBC® 056C	16 contiguous 16K byte blocks	4 blocks of 64K bytes
iSBC® 012C	32 contiguous 16K byte blocks	8 blocks of 64K bytes

NOTE: Blocks cannot cross 4M byte boundary**BASE ADDRESS****MULTIBUS®** — Any 16K byte boundary**iLBX™ BUS** — Any 64K byte boundary**Power Requirements****Voltage** — 5VDC \pm 5%

Board	Current	Standby (Battery Backup)
iSBC® 028CX	3.8A (typ.) 6.5A (max.)	2.0A (typ.) 2.2A (max.)
iSBC® 056CX	4.0A (typ.) 6.6A (max.)	2.1A (typ.) 2.3A (max.)
iSBC® 012CX	4.4A (typ.) 6.8A (max.)	2.3A (typ.) 2.5A (max.)

Environmental Requirements**Operating Temperature** — 0°C to 55°C**Operating Humidity** — To 90% without condensation**Physical Dimensions****Width** — 30.48 cm (12 inches)**Height** — 17.15 cm (6.75 inches)**Thickness** — 1.27 cm (0.50 inches)**Weight** — iSBC 028CX: 4699 gm (16.7 ounces); iSBC 056CX: 5329 gm (19.0 ounces); iSBC 012CX: 6589 gm (23.5 ounces)**Reference Manuals****145158-001** — iSBC 028CX/iSBC 056CX/iSBC 012CX Hardware Reference Manual**144456-001** — Intel iLBX Specification**9800683-03** — Intel MULTIBUS Specification

Manuals may be ordered from any Intel Sales Representative, Distributor Office or from the Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

ORDERING INFORMATION**Part Number Description**

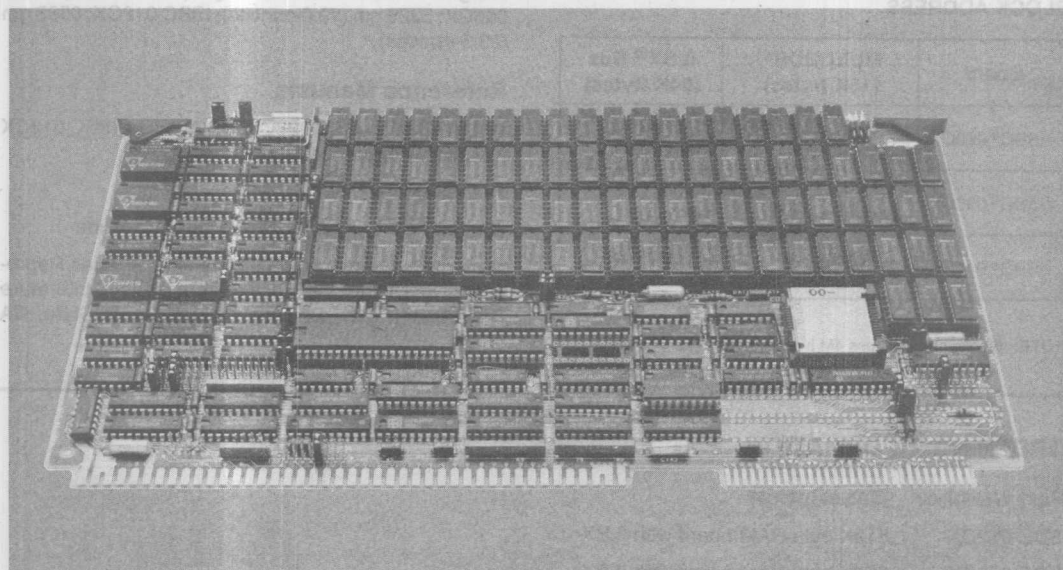
iSBC 012CX	512K byte RAM board with iLBX
iSBC 056CX	256K byte RAM board with iLBX
iSBC 028CX	128K byte RAM board with iLBX

iSBC® 028C, 056C AND 012C ECC RAM BOARDS

- iSBC® 86, iSBC® 88 RAM expansion through direct, IEEE P796, MULTIBUS® interface
- 128K, 256K, or 512K bytes of read/write memory
- Single bit error correction and double bit error detection via Intel® 8206 ECC device
- Control status register supports multiple ECC operating modes
- Error status register provides error logging by host CPU board
- Base address selectable on 16K byte boundaries
- Supports 8 or 16-bit transfer and 24-bit addressing
- Auxiliary power bus and memory protect logic for battery back-up RAM requirements

The iSBC® 028C, iSBC 056C and iSBC 012C RAM boards are members of Intel's complete line of iSBC memory and I/O Expansion boards. Each board interfaces directly to any iSBC 88 or iSBC 86 Single Board Computer via the IEEE P796 MULTIBUS® interface to expand system RAM capacity. The iSBC 028C, iSBC 056C and iSBC 012C boards contain 128K, 256K or 512K bytes of read/write memory implemented using dynamic RAM components. Single bit error correction and double bit error detection are provided on the iSBC 028C, iSBC 056C and iSBC 012C boards via the Intel 8206 Error Checking and Correction (ECC) device. Due to the on-board ECC features of the board they are ideally suited in applications where integrity of the stored data is critical, such as financial transactions, process control and medical equipment applications.

Refresh control of the RAM array is handled on-board by the RAM Array Control Logic. Therefore, no external refresh commands are necessary.



FUNCTIONAL DESCRIPTION

General

The iSBC 028C, 056C, and 012C RAM boards are physically and electrically compatible with the MULTIBUS interface standard, IEEE P796, as outlined in the Intel MULTIBUS specification. The capacity of each RAM board in this series is determined by the number of RAM devices on-board.

System Memory Size

Maximum system memory size with this series of boards is 16 megabytes. On-board jumpers assign the board to one of four 4 megabyte pages. Each page is partitioned into 256 blocks of 16K bytes each. The smallest partition on any board in this series is 16K bytes. Jumpers assign the base address (lowest 16K block) within the selected 4 megabyte page.

Error Checking and Correcting (ECC)

Error Checking and Correction is accomplished with the Intel 8206 Error Checking and Correction device. This ECC component in conjunction with the ECC check bit RAM array provides error detection and correction of single bit errors and detection only of double bit and most multiple bit errors. The ECC circuitry can be programmed to various modes to provide full diagnostic testing of both the storage and check bit RAM arrays.

ECC I/O ADDRESS SELECTION

The processor board communicates with the ECC circuitry via a single I/O port. This port is used for the Control Status Register (CSR) and the Error Status Register (ESR). The Control Status Register is programmed by the user to determine the mode of operation while the Error Status Register provides information about memory errors. The iSBC 028C, iSBC 256C and iSBC 012C RAM boards are shipped with a Programmed Array Logic (PAL) device which allows selecting one of 9 possible addresses for the I/O port. The actual selection is done by jumper configuration. Additional unprogrammed locations are left in the PAL to allow application specific I/O addresses to be defined.

CONTROL STATUS REGISTER

There are six ECC modes of operation on the "C" Series Family of RAM boards. Each mode is obtained by software programming of the CSR from the master iSBC board. The six modes are:

- a. Interrupt on any error mode
- b. Interrupt on non-correctable error only mode
- c. Correcting mode
- d. Non-correcting mode
- e. Diagnostic mode
- f. Examine syndrome word mode

Modes (a) and (b) can be used in conjunction with (c) and (d). The six modes are described below.

Interrupt on Any Error Mode — In this mode the RAM board will interrupt the iSBC processor only when any error (single or multiple bit) is detected by the ECC circuitry.

Interrupt on Non-Correctable Error Mode — In this mode the RAM board will interrupt the iSBC processor only when a non-correctable (multiple bit) error is detected by the ECC circuitry. A multiple bit error is not correctable by the ECC circuitry.

Correcting Mode — In this mode the RAM board corrects any correctable error (single-bit error). Errors which are not correctable are not modified. Interrupts are generated depending on the interrupt mode selected.

Non-Correcting Mode — In this mode the RAM board does not correct any error. The ECC circuitry continues to check for errors, but no corrective action is taken. Interrupts continue as described previously.

Diagnostic Mode — This mode is used for testing the on-board ECC circuitry. In this mode the write enable strobe to the ECC RAM array is continuously disabled. The diagnostic mode can be used to simulate errors and in conjunction with the "Examine Syndrome Word Mode" examine the check bits generated by the ECC circuitry.

Examine Syndrome Word Mode — This mode, in conjunction with the "Diagnostic Mode", is used for testing the ECC memory. In this mode, the syndrome bits/check bits are clocked into the Error Status Register (ESR) on every memory read/write cycle, respectively. The ESR translation PROM switches to a transparent mode in the Examine Syndrome Word Mode. This allows the actual syndrome word generated by the 8206 ECC device to be examined.

ERROR STATUS REGISTER

This 8-bit register contains information about memory errors. The ESR reflects the latest error occurrence. Table 1 shows the status register format. Bits 5 & 6 show the failing row while bits 0 through 4 indicate which bit (of the 16-bit data word or the 6-bit ECC syndrome word) is in error. Bit 7 is always high.

Bit		Meaning
6	5	
0	0	Error in row 0
0	1	1
1	0	2
1	1	3

Bit					Meaning
4	3	2	1	0	
0	0	0	0	0	Error in data bit 0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	Error in check bit 0
1	0	0	0	1	1
1	0	0	1	0	2
1	0	0	1	1	3
1	0	1	0	0	4
1	0	1	0	1	5
1	1	1	1	0	No Error
1	1	1	1	1	Non-correctable (multiple-bit error)

NOTE: Bit 7 is always high

Table 1.

Battery Back-up/Memory Protect

An auxiliary power bus is provided to allow separate power to the RAM array for systems requiring backup of read/write memory. An active low TTL compatible memory protect signal is brought out on the auxiliary bus connector which, when asserted, disables read/write access to the RAM board. This input is provided for the protection of RAM contents during system power-down sequences.

SPECIFICATIONS

Word Size Supported

8 or 16-bits

Memory Size

131,072 Bytes (ISBC 028C)

262,144 Bytes (ISBC 056C)

524,288 Bytes (ISBC 012C)

Access Times (All Densities)

Read/Full Write — 350 ns (max)

Write Byte — 530 ns (max)

Cycle Times (All Densities)

Read/Full Write — 460 ns (max)

Write Byte — 885 ns (max)

NOTE: If an error is detected, read access time and cycle times are extended by 255 ns.

Refresh Times

Refresh Cycle Time — 15.6 μ s

Refresh Delay Time — 760 ns

Memory Partitioning

Maximum System RAM size is 16M Bytes

PAGE ADDRESS (4M BYTES)

1 of 4 megabyte pages as follows: 0-4 megabytes; 4-8 megabytes; 8-12 megabytes; 12-16 megabytes

BLOCK ADDRESS (16K BYTES)

ISBC 028C RAM board — 8 contiguous 16K Byte Blocks (128K Bytes)

ISBC 056C RAM board — 16 contiguous 16K Byte Blocks (256K Bytes)

ISBC 012C RAM board — 32 contiguous 16K Byte Blocks (512K Bytes)

NOTE: Blocks cannot cross 4M Byte Boundary.

BASE ADDRESS

Any 16K Byte Boundary

iSBC® 028C, 056C, 012C

Power Requirements

Voltage — 5VDC \pm 5%

Current — iSBC 028C 6.5A max; iSBC 056C 6.6A max;
iSBC 012C 6.8A max

Standby — iSBC 028C 2.2A max (battery backup);
iSBC 056C 2.3A max; iSBC 012C 2.5A max

Environmental Requirements

Operating Temperature — 0°C to 55°C

Operating Humidity — To 90% without condensation

Physical Dimensions

Width — 12 inches (30.48 cm)

Height — 6.75 inches (17.15 cm)

Thickness — 0.50 inches (1.27 cm)

Weight — iSBC 028C 16.7 ounces (4699 gm); iSBC
056C 19.0 ounces (5329 gm); iSBC 012C 23.5 ounces
(6589 gm)

Reference Manuals

145183-001 — iSBC 028C/iSBC 056C/iSBC 012C Hard-
ware Reference Manual

Manuals may be ordered from any Intel Sales Represent-
ative, Distributor Office, or from the Intel Literature
Department, 3065 Bowers Avenue, Santa Clara, CA
95051.

ORDERING INFORMATION

Part Number	Description
-------------	-------------

SBC 012C	512K Byte RAM board with ECC
SBC 056C	256K Byte RAM board with ECC
SBC 028C	128K Byte RAM board with ECC

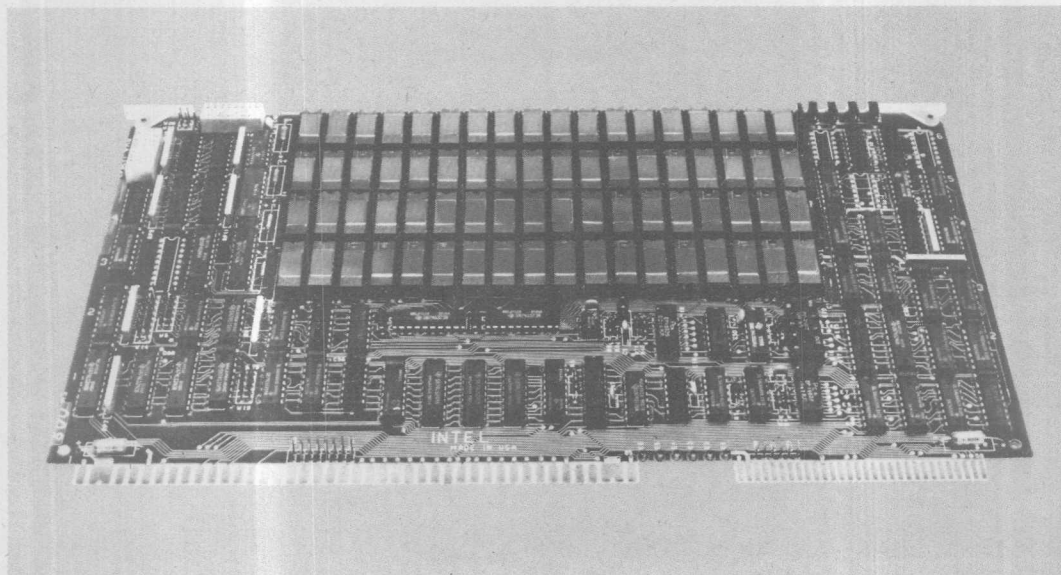


iSBC™ 012B RAM MEMORY BOARDS

- iSBC 86, iSBC 88 and iSBC 80 board RAM expansion through direct MULTIBUS® interface
- 512K of read/write memory
- On-board parity generator/checker and error status register
- Requires a single +5 volt power supply
- Assignable anywhere within a 16 megabyte address space
- Jumper selectable base address on any 16K byte boundary
- Auxiliary power bus and memory protect control logic for battery backup RAM requirements

The iSBC 012B RAM memory board is a member of Intel's complete line of iSBC memory and I/O expansion boards. The board interfaces directly to any iSBC 86, iSBC 88 or iSBC 80 Single Board Computer via the MULTIBUS interface to expand system RAM capacity. The iSBC 012B board contains 512K bytes of read/write memory implemented using dynamic RAM components. An on-board dynamic RAM controller refreshes a portion of these components every 16 microseconds. Each refresh cycle utilizes memory for 550 nanoseconds (maximum).

The iSBC 012B board generates byte oriented parity during all write operations and performs parity checking during all read operations. When a parity error is detected, the board can generate an interrupt on the MULTIBUS interface. In addition, the row and bank of the RAM array containing the error are stored in a Parity Flag Register. This register is accessible as a MULTIBUS I/O port. An on-board LED also provides a visual indication that a parity error has occurred.



The following are trademarks of Intel Corporation and may be used to describe Intel products: CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, μ Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE, ICS, iAPX and iMMX. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

SPECIFICATIONS

Word Size

8 bits and 16 bits

Memory Size

524,288 bytes (iSBC 012B)

Access Time

330 nsec (worst case)
300 nsec (typical)

Cycle Times (Worst Case)

Read — 500 ns max.
Write — 500 ns max.
Refresh — 550 ns max.

Interface

All address, data and command signals are TTL compatible.

Address Selection

Memory — Base address is jumper selectable on any 16K byte boundary in a 16 megabyte address space. On-board RAM cannot cross a 4 megabyte address boundary.

Parity Flag Register — The I/O address of the Parity Flag Register is jumper selectable to be between 00H to 0FH or 40H to 4FH.

Connector

Edge connector — 86 pin double-sided PC edge connector with 0.156 in. contact centers.

Mating connector — Viking 3KH43/9AMK12 or equivalent.

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM array for systems requiring battery backup of read/write memory. Selection of

this auxiliary RAM power bus is made via jumpers on the board.

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM on the board. This input is provided for the protection of RAM contents during system power-down sequences.

Physical Characteristics

Width — 12.00 in. (30.48 cm)
Height — 6.75 in. (17.15 cm)
Depth — 0.50 in. (1.27 cm)
Weight — 14 oz. (397 gm)

Electrical Characteristics

D.C. POWER REQUIREMENTS

All configurations require only +5 volts $\pm 5\%$.

Normal System Operation (max.)

4.8A (worst case)
3.46A (typical)

Auxiliary Power No RAM Access (max.)

1.35A (worst case)
0.88A (typical)

Environmental Characteristics

Operating Temperature — 0°C to +55°C

Relative Humidity — to 90% (without condensation)

Reference Manual

143865-001 — iSBC 056B/012B Hardware Reference Manual (not supplied)

Manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Dept., 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

SBC 012B 512K-Byte RAM Board with Parity

this auxiliary RAM power bus is made via jumpers on the board.

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM on the board. This input is provided for the protection of RAM contents during system power-down sequences.

Physical Characteristics

Width — 12.00 in. (30.48 cm)
Height — 2.75 in. (7.0 cm)
Depth — 0.50 in. (1.27 cm)
Weight — 14 oz. (397 gm)

Electrical Characteristics

D.C. POWER REQUIREMENTS

All configurations require only +5 volts \pm 5%.

Normal System Operation (max.)

4.8A (worst case)
3.48A (typical)

Auxiliary Power No RAM Access (max.)

1.38A (worst case)
0.88A (typical)

Environmental Characteristics

Operating Temperature — 0°C to +55°C

Relative Humidity — to 90% (without condensation)

Reference Manual

9001 — 18BC 000012B Hardware Reference Manual (not supplied)

Hardware may be ordered from any Intel sales office, distributor, or from Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

SPECIFICATIONS

Word Size

8 bits and 16 bits

Memory Size

524,288 bytes (18BC 012B)

Access Time

330 nsec (worst case)
300 nsec (typical)

February 1982

(Worst Case)

Read — 500 ns max.
Write — 500 ns max.
Refresh — 550 ns max.

Interface

All address, data and command signals are TTL compatible.

Address Selection

Memory — 18BC 012B is jumper selectable on any 16K pin array to 18 megabyte address space. On-board address bus is 4 megabyte address boundary.

Parity Flag Register — NO access of the Parity Flag Register is jumper selectable between 40H to 40H or 40H to 40H.

Connector

Edge connector — 68 pin double in-line connector with 0.156 in. contact center-to-center spacing.
 mating connector — Viking 3KH432B or equivalent.

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM array for systems requiring battery backup of read/write memory. Selection of

Choose the Right Level of
Memory-Error Protection

Drew Lucy
Electronic Design

ORDERING INFORMATION

Part Number Description

18BC 012B 512K-Byte RAM Board with Parity

Choose the right level of memory-error protection

With three basic methods for handling memory errors, selecting the right approach boils down not to maximizing reliability but to determining the effect and frequency of the errors.

Choosing the best approach to handling memory—particularly RAM—errors does not necessarily mean selecting the one offering the highest reliability. To minimize the impact of memory error protection on system cost and performance, the goal should be to select an approach that provides a level of reliability that meets—but does not exceed—the requirements of the application. The level of reliability required by a system should be a function of the application's sensitivity to memory errors and of the frequency with which errors will occur.

When selecting a RAM board, there are three basic error-protection approaches:

- **Unprotected memory**—designing the total system with a maximum degree of tolerance for errors in the RAM data.
- **Parity**—designing the RAM array so that it can detect when a single-bit error has occurred.
- **ECC (for error-correcting code or circuitry)**—designing the RAM array so that it not only can detect the majority of errors but also can correct single-bit errors without failing.

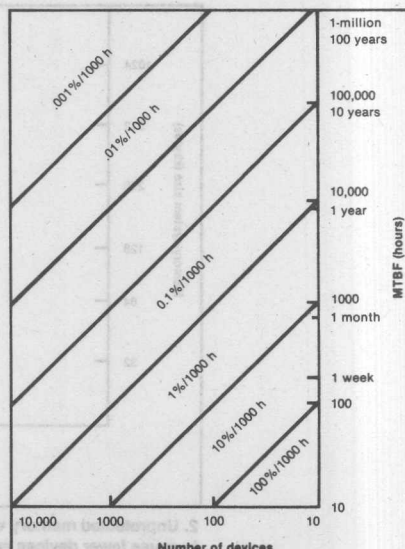
There are also three major reasons why interest in RAM reliability has grown. First, the memory arrays themselves have grown. In 1978, the typical single-board computer system had 16 to 32 kbytes of RAM, and systems with as few as 512 bytes of RAM were not uncommon. Today, the typical single-board computer system has 64 to 128 kbytes of RAM, while the largest systems have up to 512 kbytes or 1 Mbyte of RAM. Microcomputers with 16 Mbytes of RAM are not too far off. It is natural to assume that these larger RAM systems are likely to fail more often than the earlier systems, which were up to 2000 times smaller.

The second reason is a growing trend toward storing programs as well as data in dynamic RAMs instead of in ROM or EPROM. The combined hard and soft error rate for these volatile memory devices has always been higher than that of equivalently sized nonvolatile EPROMs. The consequences of an altered program instruction can be much more serious than those of an altered data byte.

The third and, perhaps the least understood, reason is Intel's discovery of alpha-particle-related soft errors, which can affect 64-kbit dynamic RAMs (see "The Rise of Alpha Particles"). These devices are essential for microcomputer memories larger than 128 kbytes to be economically and technically practical.

Failures hard and soft

When a RAM component fails to retain stored data correctly, that failure can be either "hard" or "soft."



1. A memory system's mean time between failures is a function of the number of memory devices used as well as the typical failure rates of the individual memory devices.

Drew Lucy, Memory Board Product Manager
Intel Corp.
5200 N.E. Elam Young Pkwy., Hillsboro, OR 97123

Systems & Software: Memory-error protection

Hard errors, which can affect single or multiple bits, are caused by permanent defects in the component such as shorts, open leads, microcracks, or other flaws. A hard error causes the device to consistently return the same value for the failed bit regardless of what is subsequently written to that bit.

Soft errors are random, nonrecurring, and non-destructive, and affect only a single bit. They cause the values of bits to be inverted, but do not prevent bit cells from correctly storing data from subsequent writes. Such errors can be caused by noisy system environments, poor system design, or rare combinations of noise, data patterns, and temperature that push the RAM beyond its operating limits.

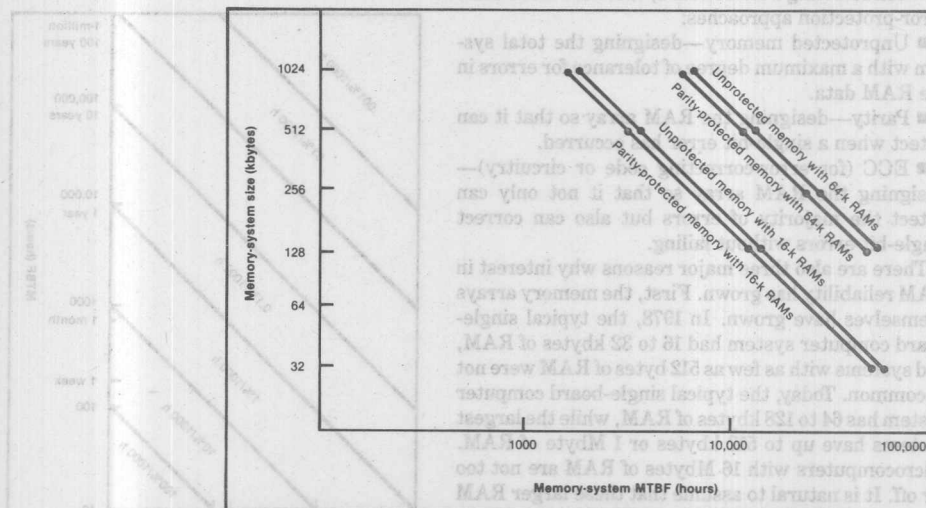
However, in a well-designed system, the primary source of soft errors is the ionizing radiation of alpha particles, which have enough energy to change the cell charge in semiconductor substrates with high impedance nodes. These helium nuclei are emitted naturally by the environment but primarily by radioactive impurities in component packaging material. As with all soft errors, alpha-particle-induced errors can be purged by rewriting the correct data to the bit cell.

During a RAM's useful life, errors occur randomly; over a given period of time, the error rate is a constant for a particular set of operating conditions (voltage, temperature, etc.). The failure rate of a RAM is expressed as the percent probability that the device will fail during a given number of hours of operation. The

mean time between failure (MTBF) for a single device is simply the reciprocal of its failure rate (a device with a failure rate of 1% per 1000 hours has an MTBF of 100,000 hours). Failure rates are additive: A component's hard and soft failure rates can be measured independently and then added together to get the device's combined failure rate. More important, the failure rate for a collection of devices assembled serially (all devices must work for the system to function) is the sum of the failure rates for the individual components.

In the case of a RAM array, in which all the devices are the same, the system failure rate is equal to the failure rate of the device times the number of devices. The MTBF of a RAM array is thus inversely proportional to the number of devices in the system—doubling the number of RAMs in the array cuts the MTBF of the system in half. (However, these relationships do not apply to memory with ECC, which is not a serial system.) Figure 1 illustrates how a RAM array's MTBF varies with the number of devices in the array for a range of device-failure rates.

The capacitors, diodes, buffers, and other support logic also contribute to reducing a memory system's MTBF. However, the failure rate for these MSI, SSI, and TTL components, whose failures are almost always hard, is quite low compared with the RAM array. For example, Intel's A-series RAM boards (iSBC 056A, iSBC 028A, etc.), which include parity, were tested at 55°C for a total of 73,200 board hours,



2. Unprotected memory will have a slightly higher MTBF than parity-protected memory, because fewer devices are required. However, note the substantial increase in MTBF gained by using 64-kbit RAMs rather than 16-k devices.

The rise of alpha particles

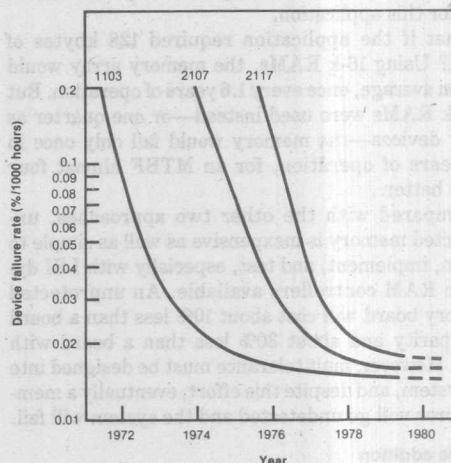
Much of the recent interest in RAM system reliability stems from concern over alpha-particle soft-error rates reported for the initial 64-k RAMs. Alpha particles affect all dynamic RAMs, but 64-k RAMs are most susceptible to alpha particles than smaller RAMs, primarily because their bit storage cells are much smaller. This size reduction reduces the amount of stored electric charge (critical charge) used to distinguish between a 1 and a 0. In the transition from 16-k to 64-k components, the critical charge was reduced to the point where a significant percentage of the alpha particles striking the storage cell contained enough energy to alter the contents of that cell.

Those early reports, in fact, indicated that the 64-k RAM was in the initial stages of a process that all new semiconductor memories go through. As semiconductor manufacturers gain experience with a new device, the failure rate drops dramatically. The figure illustrates this process for the Intel 1103 (1-k), 2107 (8-k), and 2117 (16-k) dynamic RAMs. The degree and rate of improvement in 64-k RAMs have been even more dramatic because semiconductor manufacturers were facing the unexpectedly severe alpha-particle soft-error problem for the first time.

With a potentially multibillion-dollar market at stake, the alpha-particle problem has been attacked on a number of fronts. Packaging-material manufacturers have reduced the level of radioactive thorium and uranium in their products. Meanwhile, 64-k designs have been changed to make them more immune. Different processing techniques have been developed that allow

more charge to be held in a storage cell of a given area. Finally, coatings have been developed to absorb much of the alpha particle's energy before it reaches the die.

Today, Intel 64-k dynamic RAMs have a soft-error rate of approximately 0.1% per 1000 hours and a hard-failure rate of approximately 0.02% per 1000 hours (at 55°C); the combined failure rate is 0.12% per 1000 hours. This is only 10% higher than the device failure rate for today's commercial 16-k RAMs. On a per-kilobit basis, today's 64-k RAMs are more than three and a half times as reliable as 16-k RAMs.



or 8.4 board years, without a single failure in the support logic.

Parity and ECC error protection apply primarily to the RAM components, since much of the support logic is "downstream" from where the error detection/correction takes place. If an application is so sensitive to memory errors that it must be protected from rare support-logic failures, then the design will probably require completely redundant memory systems. For the purpose of comparing error-protection approaches, this small, but not insignificant, source of memory-system failures will be ignored.

Unprotected and commonplace

The most commonly used memory for microcomputers has no inherent error protection. These systems rely on relatively high memory MTBFs and techniques like the rejection of data that fall outside predefined limits and the use of feedback from the application environment to recognize error conditions. Corrective action for these systems is limited to retrying some sequence of code and then shutting down the

system in an orderly fashion if the error condition persists.

This approach works best when RAM is used primarily for data storage. Critical code segments, including error handlers, are stored in EPROM and ROM. For example, a microcomputer-controlled lathe could regulate the feed rate of the cutting tool and the flow of lubricant being sprayed on the work piece. The desired feed rate for the material being machined and the cutting tool being used could be stored in RAM. The amount of lubricant applied could be calculated, based on a temperature input from the cutting tool.

If a RAM error causes the feed rate parameter to increase, the microcomputer may be able to detect that it exceeds some predefined limit. If not, the temperature input from the cutting tool will rise, first causing more lubricant to be applied. If the temperature input remains high, the microcomputer might revert to default feed rate stored in EPROM or execute an EPROM resident routine that retracts the tool, shuts off the lubricant, and notifies the operator. If the memory error were not detected by the pro-

gram's limit checking or temperature sensing routines, then the work piece, and possibly the cutting tool, could be ruined.

How often will this or a similar catastrophe occur? It depends on the degree of fault tolerance and the size of the memory. Assuming no fault tolerance, Fig. 2 shows how often various size RAM arrays will fail. If the lathe controller required 32 kbytes of RAM and the memory was implemented with 16-k RAMs, that RAM array could be expected to develop an error once every six and a half years of operation, on average. That is, one or perhaps two pieces would be ruined by a RAM failure during the useful life of the equipment—almost certainly an acceptable failure rate for this application.

What if the application required 128 kbytes of RAM? Using 16-k RAMs, the memory array would fail, on average, once every 1.6 years of operation. But if 64-k RAMs were used instead—or one-quarter as many devices—the memory would fail only once in 5.9 years of operation, for an MTBF almost four times better.

Compared with the other two approaches, unprotected memory is inexpensive as well as simple to design, implement, and test, especially with LSI dynamic RAM controllers available. An unprotected memory board will cost about 10% less than a board with parity and about 30% less than a board with ECC. However, fault tolerance must be designed into the system, and despite this effort, eventually a memory error will go undetected and the system will fail.

Simple addition

Parity is a simple memory-protection technique that typically appends one bit to every byte of memory. The contents of the parity bit are determined by Exclusive-ORing the data bits in the word, which results in the value of the parity bit. When the word is read, the memory system performs this operation again and checks to see that the result matches the parity bit's value. If it does not, then one bit (or any odd number of bits) in the data word has changed since the word was written. This error generates an interrupt from the memory to the single-board computer, which, in response, will usually execute an EPROM-resident shutdown or restart routine.

The system response to a parity error is the same as the system's response to an error condition detected through feedback, except that virtually all memory errors will be detected. Note that parity cannot detect a double-bit error in a data word. However, in a dynamic RAM array, the probability of two bits failing in the same word is extremely small.

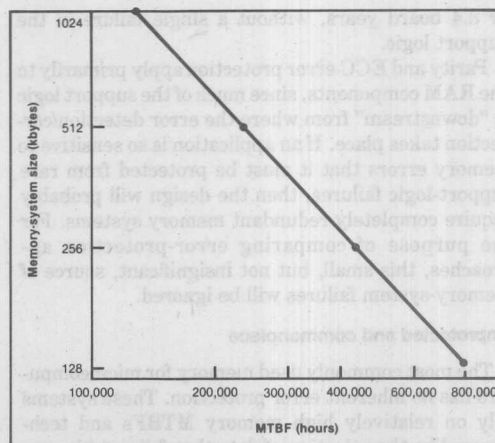
Figure 2 graphically presents the MTBFs for a variety of memory sizes with parity using both 16-k and 64-k RAMs. Once again, for any given memory

size, the 64-k-based system is more than three and a half times more reliable. A comparison of the unprotected and parity memory MTBFs in Fig. 2 reveals an interesting effect of parity. Although parity allows detection of virtually all memory errors, it also reduces the MTBF of the memory array by 11%, because the parity system requires additional memory chips, which in turn increase the system failure rate. Even so, a 256-kbyte, 64-k-RAM-based memory with parity will produce a detectable single-bit error only once every 2.6 years of operation.

The primary benefit of parity is that the system will reliably detect close to 100% of all memory errors. This allows the orderly shutdown and restarting of the microcomputer system. There are also two primary limitations: Parity produces the lowest memory MTBF of the three approaches and provides very little information as to the source or nature of the error.

Some parity boards, such as the Intel iSBC 056A and iSBC 012B, include parity-error status registers which can isolate the memory error to one quadrant of the RAM array. This information can be used by a diagnostic routine to search memory and identify the faulty memory location. If the faulty memory location is in a data area, the system may elect to continue operation. If the error occurred in an area used to store programs, or if the system cannot search memory, there is usually no alternative except to stop and reboot the system.

Whether ECC stands for error-correcting code or error-correcting circuitry, the term refers to a data-encryption scheme that appends a number of ECC



3. Failure data for various-sized memory systems using ECC protection show a substantial increase in MTBF over the parity and unprotected memories of Fig. 2. This curve was derived for a 64-kbit RAM array using 16-bit words and six ECC check bits per word.

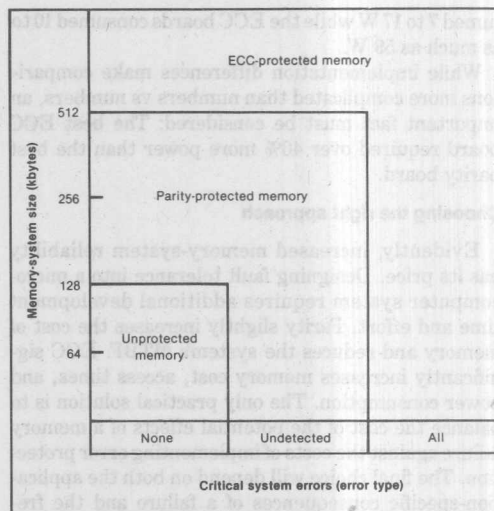
check bits to every memory word. The number of ECC bits required depends on the number of bits in the data word and on whether double-bit errors are to be detected. An 8-bit data word will require four or five ECC bits. A 16-bit word will require five or six bits and, a 32-bit word will require six or seven bits. Thus, the overhead associated with ECC grows rapidly as the data words get smaller—for a 32-bit word, the minimum overhead is 19%, but for an 8-bit word, it is 50%.

So many ECC check bits are required because they are used not only to detect errors, but also to identify which bit failed. This information allows an LSI ECC controller chip in the memory system to correct the error before passing the data to the CPU. In addition, most ECC systems can detect when two bits in the same word fail. However, standard ECC cannot correct double-bit errors. The probability of a double-bit error occurring defines the MTBF of an ECC-protected memory array.

ECC-protected memory is not a serial system. One or more elements of the system can fail without causing a system failure. This greatly complicates the calculation of system-failure probabilities. Preventive maintenance also has a significant effect on ECC memory-failure rates, as do the hard failure modes of a particular RAM device type. Figure 3 shows MTBF data for a RAM array with 16-bit words and six ECC check bits per word. (Those interested in the theory or mechanics of calculating MTBFs for ECC memory can obtain a two-part Intel application note, AP-46, AP-73; AP-73 also contains a listing of the Fortran program that was used to generate the data in Fig. 3.)

The advantages of ECC are pronounced: An ECC system's MTBF will be more than 20 times higher than the MTBF of an equivalent-sized unprotected or parity-protected memory. With preventive maintenance, ECC memory MTBFs can be over two orders of magnitude higher. With this kind of MTBF enhancement, the failure of a 1-Mbyte memory array becomes a once-in-a-lifetime event. In addition, the probability of an undetected (a triple-bit) memory error is effectively zero. However, the essentially perfect data integrity achieved with ECC brings on significant drawbacks.

For one thing, the use of ECC increases the material, assembly, and test costs of building a memory board. The major portion of the material cost increase comes from adding six ECC RAM components for every 16 data RAMs and from the ECC controller chip. Additional material costs are associated with a more complex PC board and with error logging. The net effect is an approximate 35% increase in price for a completed 16-bit RAM board. At today's prices, ECC will add about \$1000 to the list price of a 512-kbyte RAM board.



4. The best RAM-error protection for an application depends on both the memory-system size and the consequences of a single-bit memory error. The terms along the horizontal axis (none, undetected, and all) refer to the types of errors that are critical to system operation, thus indicating which error protection scheme can be used.

Another disadvantage is an increase in the access and cycle times incurred by the system to do the error checking. Depending on how the ECC is implemented, the increase in data-word access time will be between 50 and 200 ns, and the increase in cycle time will be between 100 and 300 ns. A comparison of data sheets for ten 512-kbyte Multibus RAM boards reveals parity-board access times ranging from 250 to 300 ns, while ECC-board access times ranged from 300 to 460 ns. Parity-board cycle times ranged from 400 to 500 ns; ECC-board cycle times, from 500 to 700 ns. The effect on actual system performance, of course, will depend on the system's sensitivity to memory-access time.

There is an additional performance penalty associated with accessing 8-bit data in a memory system with ECC implemented for 16-bit words. This affects both 8-bit microprocessors and 16-bit microprocessors when they operate on 8-bit data. The ECC check bits must be calculated for the full 16-bit word, which means that to write 8 bits, a 16-bit word must first be read. The new data byte is merged into the 16-bit word; then the full word is placed in memory with a new set of check bits. This read-modify-write process can double the write access time of the memory.

A third drawback of ECC is increased power consumption, stemming from the 35% increase in memory components and the additional support logic required. Comparing the Multibus 512-kbyte boards again reveals that the parity-protected boards con-

Systems & Software: Memory-error protection

sumed 7 to 17 W while the ECC boards consumed 10 to as much as 59 W.

While implementation differences make comparisons more complicated than numbers vs numbers, an important fact must be considered: The best ECC board required over 40% more power than the best parity board.

Choosing the right approach

Evidently, increased memory-system reliability has its price. Designing fault tolerance into a microcomputer system requires additional development time and effort. Parity slightly increases the cost of memory and reduces the system's MTBF. ECC significantly increases memory cost, access times, and power consumption. The only practical solution is to balance the cost of the potential effects of a memory failure against the costs of implementing error protection. The final choice will depend on both the application-specific consequences of a failure and the frequency of failure.

For many applications, as in the lathe example, the consequences of undetected memory failure may be serious, but the occurrence may be rare enough to make the costs of memory protection higher than the costs resulting from an error. Applications not seriously affected by occasional memory errors include graphics terminals, office-automation systems, speech synthesis/recognition, building-environment control, and some data-acquisition, instrumentation, and machine-control applications. In these applications, a memory error either may not significantly affect the system output or the consequences may be small and are easily correctable. For these applications, a well-designed unprotected memory board is the most cost-effective choice (Fig. 4).

If the designer concludes that the system cannot tolerate undetected errors occurring with the frequency implied by the required memory size, the next thing to do is determine the consequences of a detected memory error on the system. The impact could be significantly different from that of an undetected error. When a memory error is not detected, the system continues to run in an unknown state. In this condition, the system might cause additional damage beyond the original memory error. If the memory error can be detected, then the system can respond in a controlled fashion, thus avoiding additional system errors. If a memory error is detected (parity interrupt) in the lathe example, the microcomputer could execute an EPROM-resident error routine that would retract the cutting tool, turn off the other machinery, and trigger an alarm. The lathe could then be restarted with a minimum of downtime.

Besides most machine-control applications, parity error detection is usually sufficient for laboratory

automation, communications, and disk-based small data-processing applications. The common feature for these applications is that if a memory error can be detected immediately, the impact on the system is small; restarting the system is relatively easy. Parity is also the usual fall-back solution for applications where the consequences of an undetected memory error and the error rate combine to make unprotected memory impractical. Though it will cause the system error rate to increase slightly, parity can limit the consequences of a failure to the point of being the most cost-effective solution.

Thanks to 64-k RAMs, the maximum memory size that can be implemented with parity has increased. With 16-k RAM technology, the capacity limit was around 128 kbytes. Beyond this capacity, the number of components required reduced 16-k RAM-based system MTBFs below acceptable limits for most applications. Today, many microcomputer and minicomputer manufacturers and add-in memory vendors are producing 256- and 512-kbyte parity boards with 64-k RAMs and are achieving MTBFs comparable to older 16-k RAM-based 64-k and 128-kbyte designs.

The ability to detect single-bit errors will not provide adequate memory protection in applications where the loss of RAM data is intolerable, where the consequences of stopping the microcomputer, even in a controlled fashion, are prohibitively expensive, or where the memory is so large that the device count, even with 64-k RAMs, produces an unacceptably low MTBF.

In all these applications, the system being controlled and monitored by the microcomputer will continue to operate whether or not the computer is controlling it. In these critical situations, reliability is of primary importance and the cost of using ECC-protected memory is more than offset by the up to two-orders-of-magnitude increase in memory system MTBF.

ECC is also desirable for almost all applications requiring more than 512 kbytes of RAM (Fig. 4). The MTBF of a 512-kbyte memory with parity is about 16 months of operation. If the system runs eight hours a day, the MTBF will be almost four calendar years, which is acceptable for many but not all noncritical applications. Above 512 kbytes, the MTBF of parity memory quickly drops below one year and a 1-Mbyte memory will have an MTBF of less than eight months. For these large systems, which are appearing in greater numbers, ECC-protected memory is standard in almost all cases. □

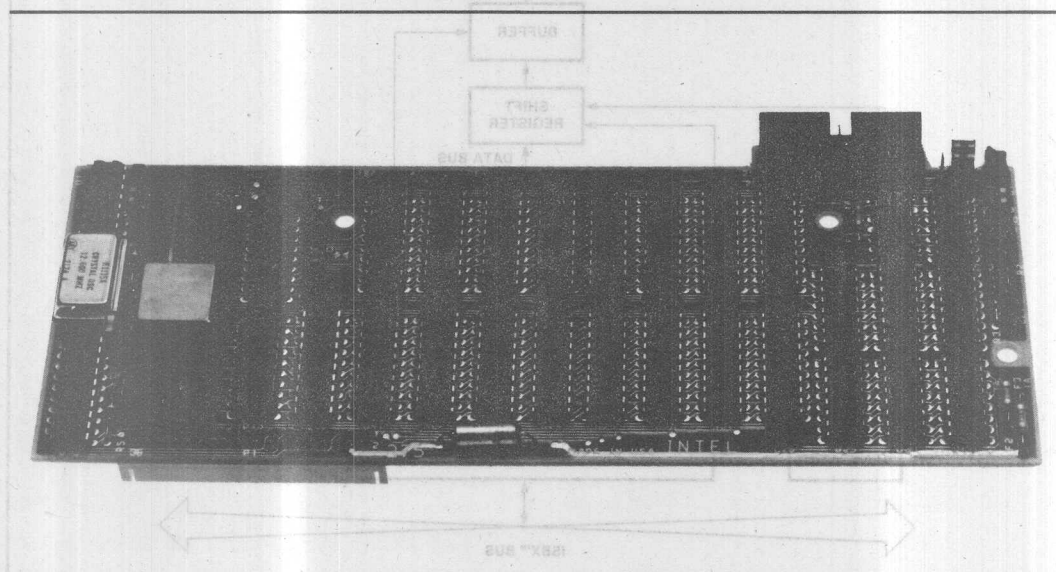
iSBX™ 275 VIDEO GRAPHICS CONTROLLER

- Complete video graphics display controller on an iSBX™ MULTIMODULE™ board
- Interfaces to either black and white or color raster scan display monitors
- 50 Hz or 60 Hz frame rate operation
- On-board refresh memory supports 512 × 512 black and white or 256 × 256 eight color display resolution
- High level drawing commands include line, arc, circle, rectangle, character, area fill, pan and scroll
- Includes Intel's 82720 Graphic Display Controller
- Compatible with industry standard iSBX™ bus interface
- Light pen interface

The iSBX 275 Video Graphics Controller (VGC) allows the user to add high level video display capability to his/her computer system with minimal cost and effort. The iSBX 275 module provides a completely self-contained bit-mapped graphics subsystem on a 3" × 7" iSBX MULTIMODULE board. This same subsystem supports either black and white or eight color displays.

In addition, iSBX 275 VGC off-loads the system CPU from many of the graphics drawing functions. Under the control of the Intel 82720 Graphics Display Controller (GDC), the iSBX 275 board directly supports high level drawing commands which includes lines, arcs, circles, rectangles, characters, area fill, pan and scroll.

The iSBX 275 MULTIMODULE board is compatible with any computer board or system product supporting the industry standard iSBX bus; this includes most board and system products from Intel. Applications for the iSBX 275 VGC include video displays for industrial operator stations, engineering work stations, videotex, business presentation systems and other information display systems.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL, MULTIMODULE and ICS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

iSBX™ Interface

The iSBX 275 VGC communicates with the host board through the iSBX bus. The iSBX bus is a standard I/O expansion bus interface (mechanical and electrical) for any microprocessor system. The iSBX standard interface allows system designers to optionally add incremental I/O functionality after the host microprocessor architecture is complete. In the case of the iSBX 275 VGC, the host board passes commands, data and status to and from the 82720 controller via two iSBX bus I/O ports.

The software interface consists of a series of high level commands passed to the 82720 controller. Table 1 contains a summary of 82720 software commands.

CRT Controller

The Intel 82720 is an intelligent graphics controller designed to be the heart of a raster-scan computer graphics display system. The 82720 performs all the basic timing needed to generate the raster display and manage the display memory. In addition, the 82720 supports several high level graphics figure drawing functions.

Table 2 lists several CRT vendors compatible with the iSBX 275 VGC.

Display Screen

The iSBX 275 VGC contains 32K bytes of high speed display memory, all of which is under the control of the 82720. The 82720 takes care of both writing and reading data to and from the screen and refreshing the screen.

The on-board display memory is organized as 16K words of 16-bits each. The 82720 reads or writes 16-bits of display data at a time. When displaying, the 82720 starts at the top left hand corner of the screen and sequences down the screen toward the bottom right hand corner.

In B&W mode all 16K, 16-bit words are treated as a contiguous block of memory, where a logical "1" in memory is displayed as an illuminated pixel.

In the color mode, three color planes, Red, Blue and Green, exist sequentially in memory but are displayed simultaneously. Each plane consists of 4K, 16-bit words where a logical "1" in a plane illuminates the corresponding color in that particular pixel.

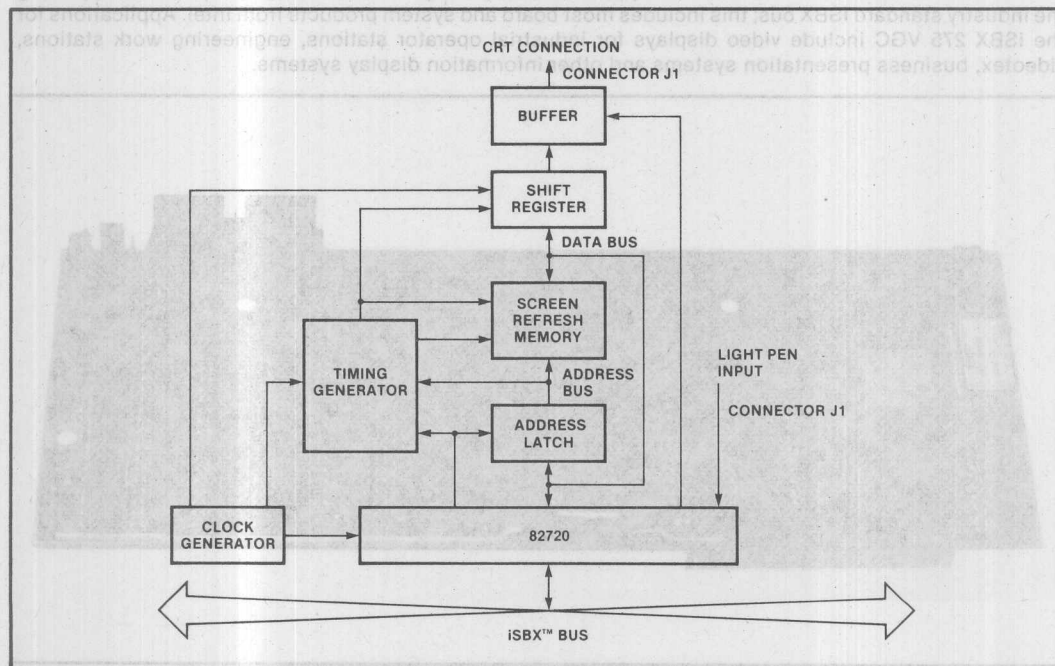


Figure 1. iSBX™ 275 VGC Block Diagram

Table 1. 82720 Command Summary

Video Control Commands	
RESET:	Resets the GDC to its idle state.
SYNC:	Specifies the video display format.
CCHAR:	Specifies the cursor and character row heights.
Display Control Commands	
START:	Ends idle mode and unblanks the display.
BCTRL:	Controls the blanking and unblanking of the display.
ZOOM:	Specifies zoom factors for graphics character writing.
CURS:	Sets the position of the cursor in display memory.
PRAM:	Defines starting addresses and lengths of the display areas and specifies the eight bytes for the graphics character.
PITCH:	Specifies the width of the X dimension of display memory.
Drawing Control Commands	
WDAT:	Writes data words or bytes into display memory.
MASK:	Sets the mask register contents.
FIGS:	Specifies the parameters for the drawing processor.
FIGD:	Draws the figure as specified above.
GCHRD:	Draws the graphics character into display memory.
Data Read Commands	
RDAT:	Reads data words or bytes from display memory.
CURD:	Reads the cursor position.
LPRD:	Reads the light pen address.

Table 2. CRT's (B&W and Color)¹

Type	Vendor	Model #
B&W	Ball Brothers	TTL 120
	Motorola	M3570
	TSI	MDC-15
Color	Ball Brothers	7-015-0131
	IDT	19AC
	CONRAC	5711C13
	HITACHI	HM-2719/2713, HM-1719/1713
	NEC	1202DH
	MITSUBISHI	C-3419

¹NOTE: This in no way constitutes an endorsement by Intel Corporation of these companies' products.

CRT Interface

The iSBX 275 VGC will interface to many B&W and RGB (Red, Green and Blue) color display monitors. For B&W monitors, the iSBX 275 board provides TTL level signals for video, vertical sync and horizontal sync or combined sync. When operating in the color mode, the iSBX 275 module provides TTL level 75 ohm line drivers for Red, Green, and Blue Video and a combined sync allowing 8 different colors to be displayed.

Composite video is not provided on the iSBX 275 MULTIMODULE board; however, with minimal external circuitry, composite video can be added (sample composite video circuit designs are included in the iSBX 275 Hardware Reference Manual).

Light Pen Interface

Light pen I/O devices may be directly interfaced to the iSBX 275 VGC. A light pen input or "hit" is triggered on the rising edge of the light pen signal and is indicated by a status bit in the 82720. The memory address of the light pen hit is obtained with a LPRD (Light Pen Read) command.

Table 3 lists a light pen vendor whose product interfaces to the iSBX 275 VGC.

Table 3. Light Pens¹

Vendor	Model #
Information Control Co.	LP-700

¹NOTE: This in no way constitutes an endorsement by Intel Corporation of this company's products.

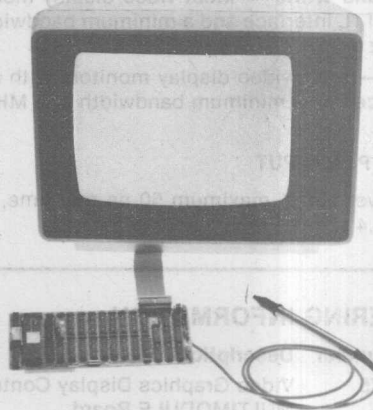


Figure 2. The iSBX™ 275 VGC Interfaces to a User-Supplied Video CRT and Light Pen

SPECIFICATIONS

Controller Characteristics

DISPLAY RESOLUTION

Black and White — nominal 512 × 512 × 1, interlaced

Color — nominal 256 × 256 × 3, non-interlaced

CRT OUTPUTS

Black and White — TTL level Video, HSYNC, VSYNC or CSYNC; maximum dot rate 13 MHz

Color — TTL level, 75 ohm line drivers for RGB and combined sync provide 8 different display colors with a 9.75 MHz maximum dot rate

FRAME RATE

50 Hz or 60 Hz via programmable option (non-interlaced)

VIDEO CONTROL

Pan and user selectable display and background color

DRAWING CONTROL

Lines, arcs, circles, rectangles, characters and area fill

CHARACTERS

Any user defined 8 × 8 font

MONITOR

Black and White — Most video display monitors with a TTL interface and a minimum bandwidth of 12 MHz

Color — Most video display monitors with a TTL interface and a minimum bandwidth of 6 MHz

LIGHT PEN INPUT

TTL level pulse, maximum 50 ns rise time, minimum 1.4 μ S hold time

ORDERING INFORMATION

Part Number	Description
SBX 275	Video Graphics Display Controller MULTIMODULE Board

Compatibility

CPU

Any iSBC single board computer or I/O board compatible with the MULTIBUS system bus and implementing the iSBX bus and connector

Physical Characteristics

Width — 3.08 inches (7.82 cm)

Height — 0.8 inches (2.05 cm)

Length — 7.5 inches (19.05 cm)

Shipping Weight — 0.5 pounds (0.175 Kg)

Mounting — Occupies one double-wide iSBX MULTIMODULE position on boards; increases board height (host plus iSBX board) to 1.14 inches (2.90 cm)

Electrical Characteristics

Power Requirements — +5 Vdc @ 1.5A

Environmental Characteristics

Temperature — 0° to 55°C (operating); -55°C to +85°C (non-operating)

Humidity — Up to 90% relative humidity without condensation (operating); all conditions without condensation or frost (non-operating)

Equipment Supplied

iSBX 275 VGC Controller

Reference Schematic — Cabling and connectors from the VGC controller to the CRT and light pen are not supplied with the controller. Cables can be fabricated with commercially available cable and connectors as described in the iSBX 275 Hardware Reference Manual.

Reference Manual

144829-001 — iSBX 275 Video Graphics Display Controller Hardware Reference Manual (NOT SUPPLIED)

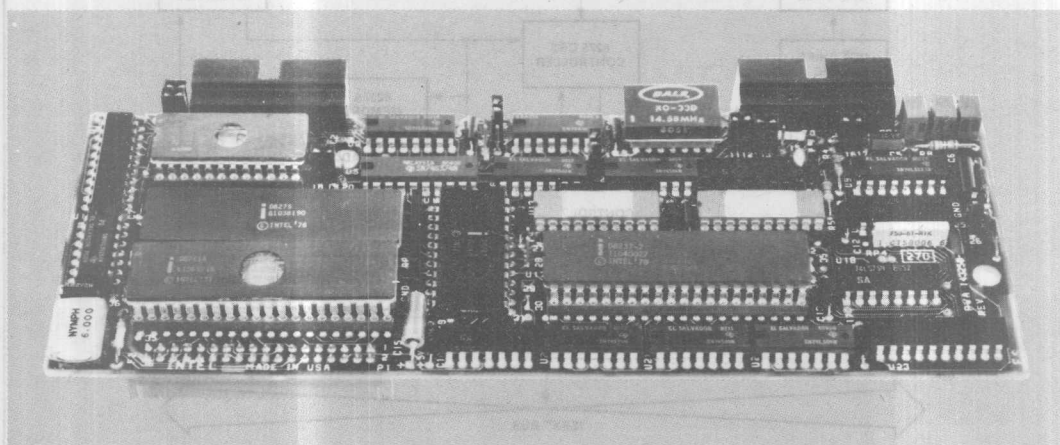
Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

iSBX™ 270 VIDEO DISPLAY CONTROLLER

- Complete video display controller on a double-wide iSBX™ MULTIMODULE™ board
- Interfaces to either black and white or color display monitors
- Displays 7 × 9, 5 × 7 or 6 × 8 character fonts
- High level software interface via a pre-programmed 8041A UPI
- Interchangeable character fonts available in EPROM
- Keyboard and light pen interface provided on-board
- 50 Hz or 60 Hz frame rate operation
- Provides cursor control, reverse video, blinking, underline, highlight and page or scroll mode
- Compatible with all 8/16 bit iSBC™ boards which support the Intel iSBX™ bus
- Graphics capability via pre-defined graphic character fonts

The iSBX 270 Video Display Controller (VDC) is a complete video controller on a standard double wide Intel iSBX MULTIMODULE board. Providing either black and white (B&W) or eight-color displays, the iSBX 270 VDC brings alphanumeric video control to the iSBX bus. Any computer board or system supporting the Intel iSBX MULTIMODULE bus is compatible with the iSBX 270 VDC, including most board and system products from Intel. Additionally, the iSBX 270 VDC supports keyboard and light pen I/O on-board; this simplifies the design of intelligent terminals.

The iSBX 270 module allows the user to add high level video display capability to his/her computer system with a minimal cost and effort. Typical applications for the iSBX 270 VDC include video displays for industrial operator stations, word processing systems, data base management products and many other uses.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

FUNCTIONAL DESCRIPTION

iSBX™ Interface

The iSBX 270 VDC interfaces to the Intel iSBX bus via the 8041A Universal Peripheral Interface (UPI) Microcomputer. The 8041A, under firmware control, provides communication between the base board and the iSBX 270 controller circuitry via the iSBX data and control lines. Data may be displayed immediately following power up, using default initialization provided by the 8041A UPI. In addition, eight high-level commands are provided by the iSBX 270 firmware; these eight commands are used to alter the default initialization of the controller and determine status. Following initialization, characters are displayed on the CRT by simply writing to the proper I/O port.

CRT Interface

The iSBX 270 VDC will interface to many B&W and RGB color display monitors. For B&W monitors, the iSBX 270 board provides TTL level signals for video, vertical sync and horizontal sync. Additionally, in B&W, two levels of intensity (normal and highlight) are supported under program control.

When operating in the color mode, the iSBX 270 module provides TTL level 75 ohm line drivers for Red, Green, and Blue Video and sync allowing 8 different colors to be displayed.

Composite video is not provided on the iSBX 270 MULTIMODULE board; however, with minimal external circuitry, composite video can be added (circuit design available; contact the local Intel Sales Office for details).

Table 1 lists several CRT vendors compatible with the iSBX 270 VDC.

Table 1. CRT's (B&W and Color)¹

TYPE	VENDOR	MODEL #
B&W	Ball Brothers	TTL 120, TV 120, TV 50
	Motorola	M3570
	TSD	MDC-15
	ELSTON	DM30-12B0-51-A04
Color	Ball Brothers	7-015-0131
	IDT	19AC
	CONRAC	5711C13
	NEC	1202DH
	MITSUBISHI	C-3419

¹NOTE: This in no way constitutes an endorsement by Intel Corporation of these companies' products. The companies listed are known to provide products compatible with the iSBX 270 board.

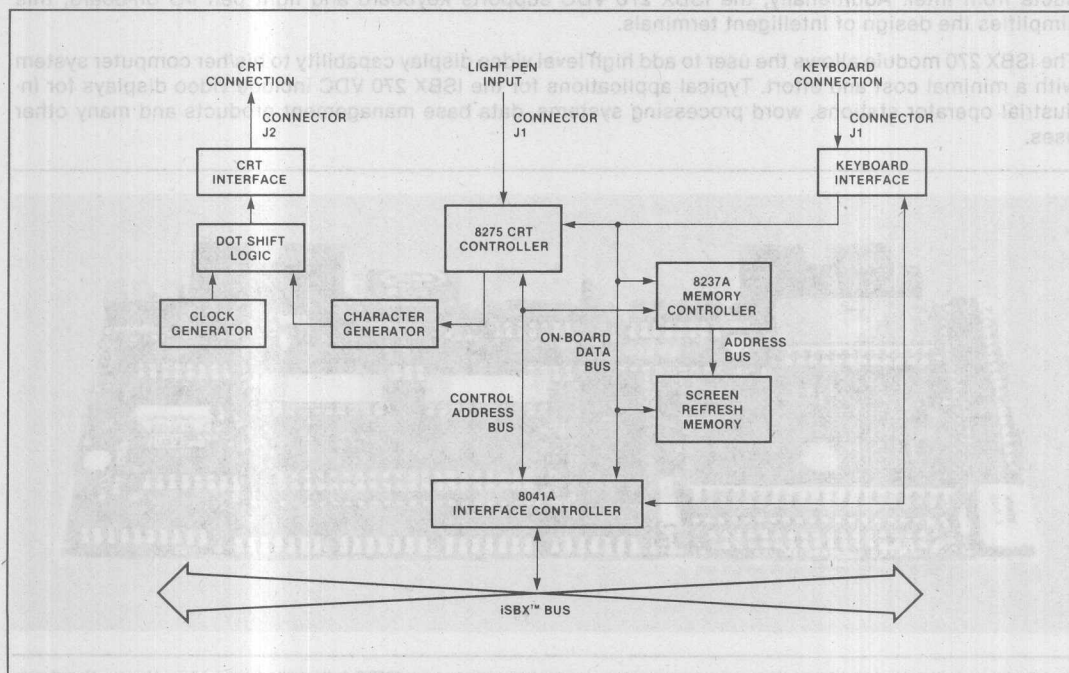


Figure 1. iSBX™ 270 VDC Block Diagram

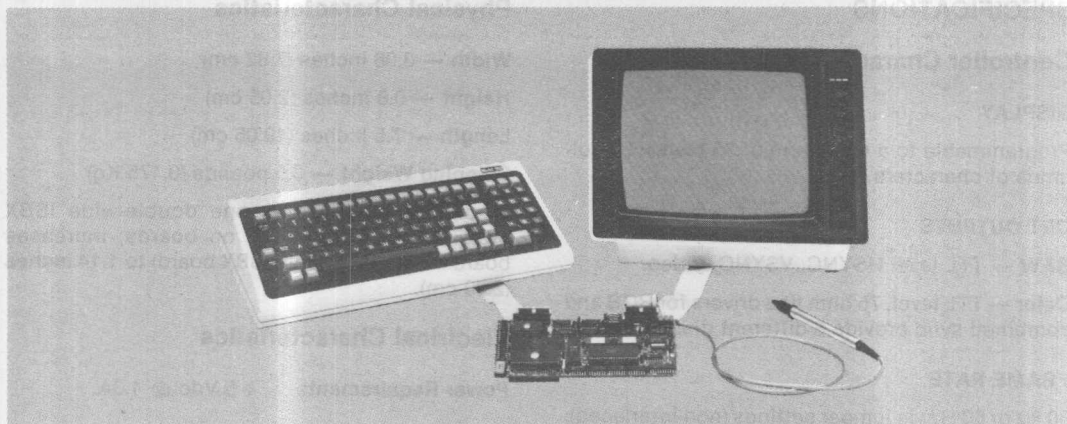


Figure 2. The iSBX™ 270 VDC Interfaces to a User-Supplied Video CRT, Keyboard and Light Pen

CRT Controller

The CRT Controller performs all timing and data buffering functions for the CRT. The iSBX 270 VDC uses the Intel 8275 CRT Controller (for additional details refer to the 8275 data sheet available from Intel).

Screen Refresh

The iSBX 270 VDC contains 4K bytes of high speed static RAM, as well as a high speed DMA controller (8237A). The 8237A, under the control of the 8041A UPI, takes care of both writing data to the screen and refreshing the screen.

Character Generation

The character fonts (128 characters, including alphabetic, numeric, and special characters) that are displayed on the CRT are stored in EPROM. The need may arise to display different character fonts, i.e., those used in international systems or custom symbols which are application specific. With the iSBX 270 VDC the user may modify any or all of the character fonts by simply reprogramming the EPROM. In addition, the user may utilize a larger EPROM to obtain up to 256 characters.

Keyboard Interface

The iSBX 270 VDC also interfaces to a keyboard I/O device via the J1 edge connector. The keyboard interface of the iSBX 270 VDC accepts up to eight TTL parallel data lines and one TTL strobe, either positive or negative. Keyboard input is indicated by a status bit in the 8041A and/or an interrupt. In addition, control lines are provided for visual and/or audible indicators.

Table 2 lists several keyboards that interface to the iSBX 270 VDC.

Table 2. Keyboards¹

VENDOR	MODEL #
Advanced Input Devices	SK-067
Cherry	B70-05AB
Cherry	CB80-07AA
Chomerics	AN26109/AE26203
Cortron	35-500014
Keytronic	L1648
Keytronic	L1660
Keytronic	L1674-03
Keytronic	L1752
Microswitch	66SD6-7
Microswitch	87SD30-8

¹NOTE: This in no way constitutes an endorsement by Intel Corporation of these companies' products. The companies listed are known to provide products compatible with the iSBX 270 board.

Light Pen Interface

Light pen I/O devices may be directly interfaced to the iSBX 270 VDC. A light pen hit is triggered on the rising edge of the light pen signal and is indicated by a status bit in the UPI 8041A and/or an interrupt.

Table 3 lists a light pen vendor whose product interfaces to the iSBX 270 VDC.

Table 3. Light Pens¹

VENDOR	MODEL #
Information Control Co.	LP-700

¹NOTE: This in no way constitutes an endorsement by Intel Corporation of this company's products. The company listed is known to provide products compatible with the iSBX 270 board.

SPECIFICATIONS

Controller Characteristics

DISPLAY

Programmable to a maximum of 35 rows x 80 columns of characters.

CRT OUTPUTS

B&W — TTL level HSYNC, VSYNC, Video.

Color — TTL level, 75 ohm line drivers for RGB and combined sync provide 8 different display colors.

FRAME RATE

50 Hz or 60 Hz via jumper settings (non-interlaced).

CHARACTER FONTS

5x7, 7x9 or 6x8 jumperable with appropriate crystal. Character generator uses 2716 EPROM. Also compatible with 2732A EPROM's. For generation of special fonts, please refer to iSBX 270 VDC Hardware Reference Manual.

VIDEO CONTROL

Reverse video, blinking, underline, highlight, cursor control and page or scroll mode.

TV MONITOR

Most video display monitors with a 10 MHz bandwidth or better.

LIGHT PEN INPUT

TTL level pulse, maximum 50 ns rise time, minimum 100 ns hold time.

Compatibility

CPU

Any iSBC single board computer or I/O board compatible with the MULTIBUS system bus and implementing the iSBX bus and connector.

ORDERING INFORMATION

Part Number Description

SBX 270	Video Display Controller MULTIMODULE Board
---------	---

Physical Characteristics

Width — 3.08 inches (7.82 cm)

Height — 0.8 inches (2.05 cm)

Length — 7.5 inches (19.05 cm)

Shipping Weight — 0.5 pounds (0.175 Kg)

Mounting — Occupies one double-wide iSBX MULTIMODULE position on boards; increases board height (host plus iSBX board) to 1.14 inches (2.90 cm).

Electrical Characteristics

Power Requirements +5 Vdc @ 1.3A.

Environmental Characteristics

Temperature — 0°C to 55°C (operating); -55°C to +85°C (non-operating).

Humidity — Up to 90% relative humidity without condensation (operating); all conditions without condensation or frost (non-operating).

Equipment Supplied

iSBX 270 VDC Controller
Reference Schematic

Cabling and connectors from the VDC controller to the CRT, keyboard and light pen are not supplied with the controller. Cables can be fabricated with commercially available cable and connectors as described in the iSBX 270 Hardware Reference Manual.

Reference Manual

143444-001 — iSBX 270 Video Display Controller Hardware Reference Manual (NOT SUPPLIED).

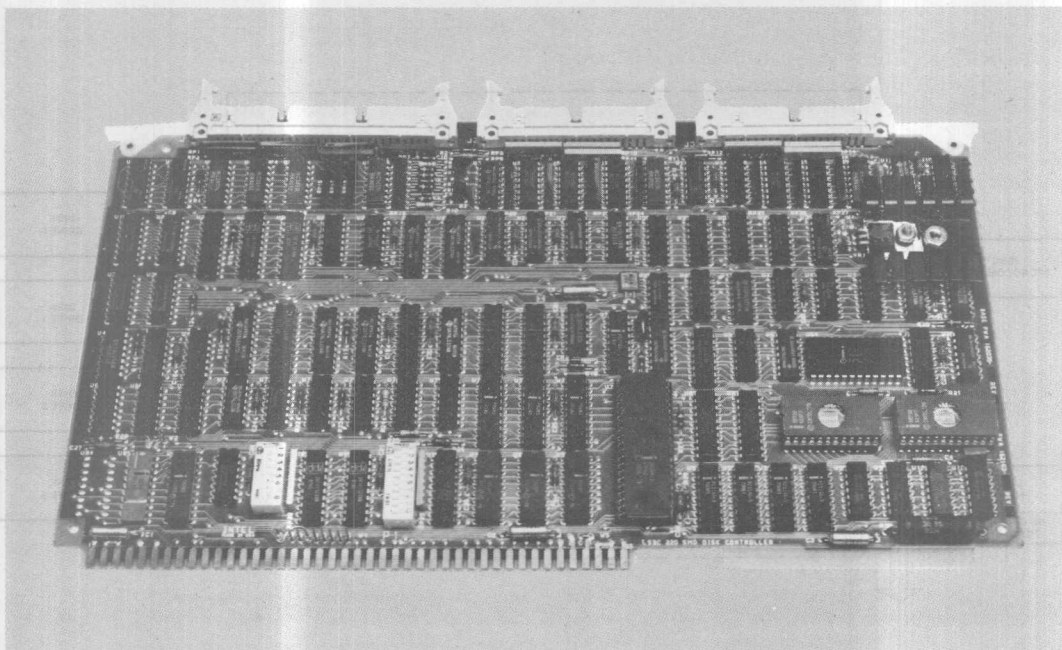
Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.



iSBC™ 220 SMD DISK CONTROLLER

- Controls up to four SMD interface compatible disk drives
- 12 MB to 2.4 GB per controller
- Compatible with all iSBC™ 80, iSBC™ 88, and iSBC™ 86 Single Board Computers
- Intel® 8089 I/O Processor provides two high speed DMA channels as well as controller intelligence
- Software drivers available for iRMX™ 86 and iRMX™ 88 operating systems
- On-board diagnostic and ECC
- Full sector buffering on-board
- Capable of addressing 1 MB of system memory
- SMD interface available on 14" Winchester, CMD, SMD and large fixed-media drives

The iSBC 220 SMD Disk Controller brings very large mass storage capabilities to any iSBC 80, iSBC 88, or iSBC 86 MULTIBUS system. The controller will interface to any disk drive conforming to the industry standard SMD interface. Using simplified cable connections, up to four drives may be connected to the iSBC 220 Controller Board to give a total maximum capacity of 2.4 gigabytes. The Intel 8089 I/O Processor simplifies programming through the use of memory-based parameter blocks. A linked list technique allows the user to perform multiple disk operations.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: CREDIT, Index, Intel, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and ICS, and the combination of MCS, ICE, iSBC, iSBX or ICS, and a numerical suffix. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1980
AFN-01794A

FUNCTIONAL DESCRIPTION

Full On-Board Buffer

The iSBC 220 SMD Controller contains enough on-board RAM for one full sector buffering. The controller is designed to make use of this buffer in all transfers. The on-board sector buffer prevents data overrun errors and allows the iSBC 220 SMD Controller to occupy any priority slot on the MULTIBUS.

ECC

High data integrity is provided by on-board Error Checking Code (ECC) logic. When writing sector ID or data fields, a 32-bit Fire code, for burst error correction, is appended to the field by the controller. During a Read operation, the same logic regenerates the ECC polynomial and compares this second polynomial to the appended ECC. The ECC logic can detect an erroneous data burst up

to 32 bits in length and using an 8089 algorithm can correct an erroneous burst up to 11 bits in length.

SMD Interface

High speed, reliable data transfers are a major benefit of using the SMD interface. A data transfer rate of 1.2 MB is accomplished by using separate (radial) differential data line cabling for each drive. Control signals are daisy-chained from drive to drive.

Defective Track Handling

When a track is deemed defective, the host processor reformats the track, giving it a defective track code and enters the address of the next available alternate track. When the controller accesses a track previously marked defective, the controller automatically seeks to the assigned alternate track. The alternate track seek is totally automatic and invisible to the user.

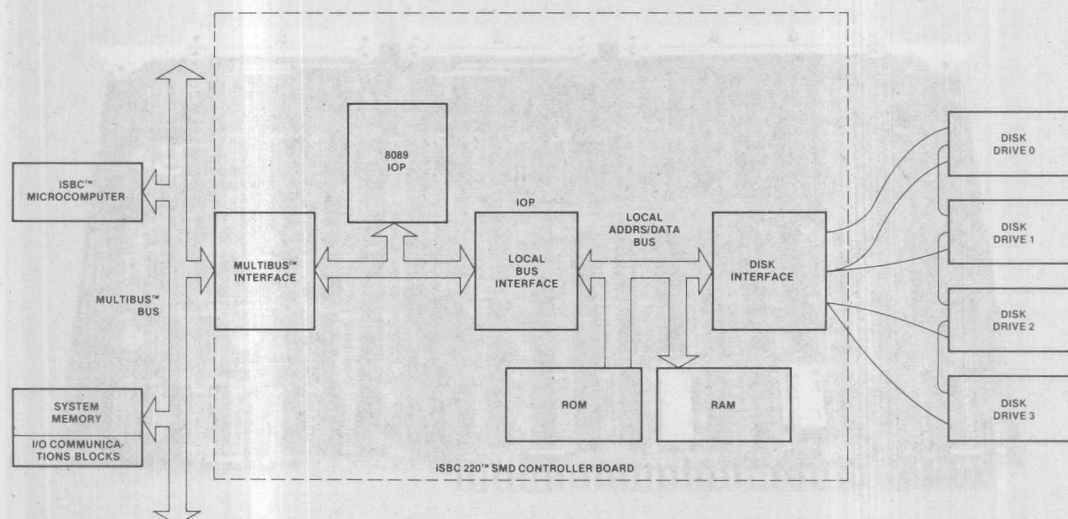


Figure 1. Simplified Block Diagram of iSBC 220™ SMD Disk Controller

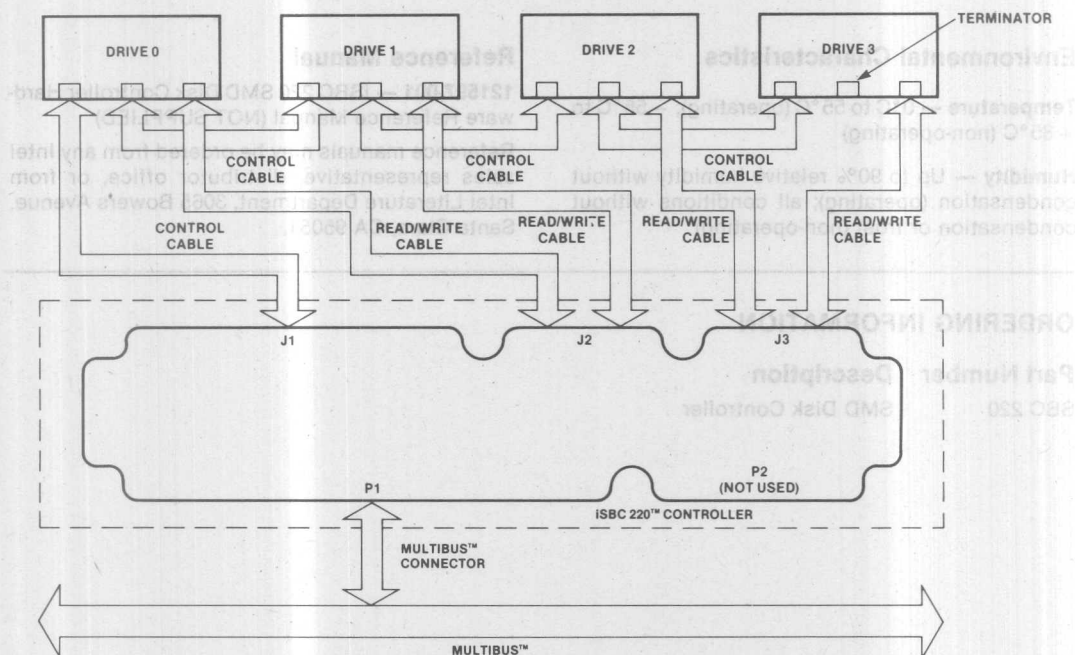


Figure 2. Typical Multiple Drive System

SPECIFICATIONS

Compatibility

CPU — Any iSBC MULTIBUS computer or system mainframe

Disk Drive — Any SMD interface-compatible disk drive

Equipment Supplied

iSBC 220 SMD Disk Controller
Reference schematic

Controller-to-drive cabling and connectors are not supplied with the controller. Cables can be fabricated with flat cable and commercially-available connectors as described in the iSBC 220 SMD Disk Controller Hardware Reference Manual.

Physical Characteristics

Width — 6.75 in. (17.15 cm)

Height — 0.5 in. (1.27 cm)

Length — 12.0 in. (30.48 cm)

Shipping Weight — 19 oz (0.54 kg)

Mounting — Occupies one slot of iSBC system chassis or cardcage/backplane

Electrical Characteristics

Power Requirements

+ 5 VDC @ 3.25A max

– 5 VDC @ 0.75A max¹

Note 1: On-board voltage regulator allows optional – 12 VDC usage from MULTIBUS.

Data Organization and Capacity

Bytes per Sector² — 128 256 521 1024

Sectors per Track² — 108 64 35 18

Note 2: Software selectable.

Table 1. Drive Characteristics (Typical)

Disk (spindle) Speed	3600 rpm
Tracks per Surface	823
Head Positioning	Closed loop servo type, track following
Access Time	Track to Track 6 ms Average 30 ms Maximum 55 ms
Data Transfer Rate	1.2 megabytes/second
Storage Capacity	12 to 2.4 gigabytes

Environmental Characteristics

Temperature — 0°C to 55°C (operating); -55°C to +85°C (non-operating)

Humidity — Up to 90% relative humidity without condensation (operating); all conditions without condensation or frost (non-operating)

Reference Manual

121597-001 — iSBC 220 SMD Disk Controller Hardware Reference Manual (NOT SUPPLIED)

Reference manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

ORDERING INFORMATION

Part Number	Description
SBC 220	SMD Disk Controller

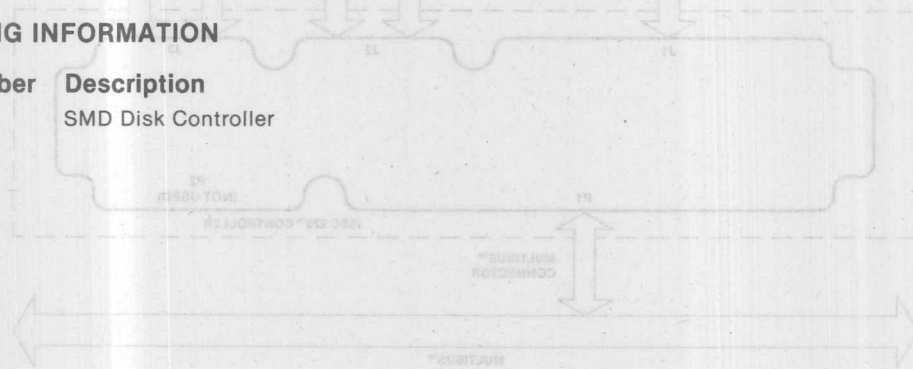


Figure 2. Typical Multiple Drive System

Electrical Characteristics
Power Requirements
+5 VDC @ 3.25A max
-5 VDC @ 0.75A max
Note 1: On-board voltage regulator allows optional -12 VDC usage from MULTIBUS.
Data Organization and Capacity
Bytes per Sector — 128 528 528 1024
Sectors per Track — 128 64 32 16
Note 2: Software selectable

Table 1. Drive Characteristics (Typical)

Storage Capacity	Data Transfer Rate	Access Time	Head Positioning	Tracks per Surface	Disk (spindle) Speed
12 to 32 gigabytes	1.2 megabytes/second	Maximum 55 ms	Average 30 ms	Track to Track 8 ms	Following
					Closed loop servo type, track
					820
					3600 rpm

SPECIFICATIONS

Compatibility
CPU — Any iSBC MULTIBUS computer or system
Disk Drive — Any SMD interface-compatible disk drive

Equipment Supplied
iSBC 220 SMD Disk Controller
Reference schematic
Controller-to-drive cabling and connector are not supplied with the controller. Cables can be fabricated with flat cables and commercially-available connectors as described in the iSBC 220 SMD Disk Controller Hardware Reference Manual.

Physical Characteristics
Width — 6.75 in. (17.15 cm)
Height — 0.5 in. (1.27 cm)
Length — 12.0 in. (30.48 cm)
Shipping Weight — 10 oz (0.34 kg)
Mounting — Occupies one slot of iSBC system chassis or cardcage backplane



ISBX 218 FLEXIBLE DISK CONTROLLER

- **ISBX MULTIMODULE controller provides flexibility at low cost**
- **Controls most single and double density diskette drives**
- **User-programmable drive parameters allow wide choice of drives**
- **Phase lock loop data separator assures maximum data integrity**
- **Read and write on single or multiple sectors**
- **Single +5V supply**

The Intel ISBX 218 Flexible Disk Controller is a double-wide iSBX board diskette controller capable of supporting virtually any soft-sectored, double density or single density diskette drive. The standard controller can control up to four drives with up to eight surfaces. In addition to the standard IBM 3740 formats and IBM System 34 formats, the controller supports sector lengths of up to 8192 bytes. The iSBX 218 board's wide range of drive compatibility is achieved without compromising performance. The operating characteristics are specified under user program control. The controller can read, write, verify, and search either single or multiple sectors.

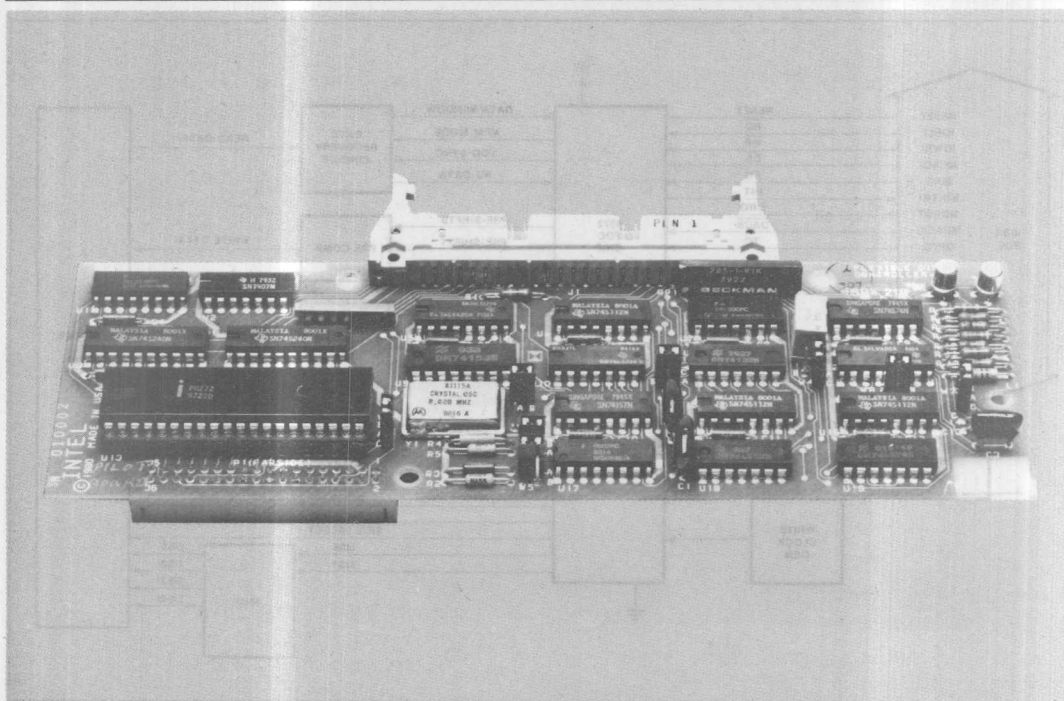


Figure 1. Block Diagram of ISBX 218 Board

FUNCTIONAL DESCRIPTION

Intel's 8272 Floppy Disk Controller (FDC) chip is the heart of the iSBX 218 Controller. On-board data separation logic performs standard MFM (double density) and FM (single density) encoding and decoding, eliminating the need for external separation circuitry at the drive. Data transfers between the controller and memory are managed by the intelligent device (usually an Intel 8-bit or 16-bit CPU chip) on the host board. A block diagram of the iSBX 218 Controller is shown in Figure 1.

Universal Drive and iSBX 218 Controller

Because the iSBX 218 Controller has universal drive compatibility, it can be used to control virtually any standard- or mini-sized diskette drive. Moreover, the iSBX 218 Controller fully supports the iSBX bus and can be used with any single board computer which furnishes this bus. Because the iSBX 218 Controller is programmable, its performance is not compromised by its universal drive compatibility. The track-to-track access, head-load, and head-unload characteristics of the selected drive model are program

specified. Data may be organized in sectors up to 8192 bytes in length.

Interface Characteristics

The standard iSBX 218 Controller includes an Intel 8272 Floppy Disk Controller chip which supports up to four drives, single or double sided.

SIMPLIFIED INTERFACE—The cables between the iSBX 218 Controller and the drive(s) may be low cost, flat ribbon cable with mass termination connectors. The mechanical interface to the board is a right-angle header with locking tabs for security of connection.

PROGRAMMING — The powerful 8272 FDC circuit is capable of executing high-level commands that simplify system software development. The device can read and write both single and multiple sectors. CRC characters are generated and checked automatically. Recording density is selected at each Read and Write to support the industry standard technique of recording basic media information on Track 0 of Side 0 in single density, and then switching to double density (if necessary) for operations on other tracks.

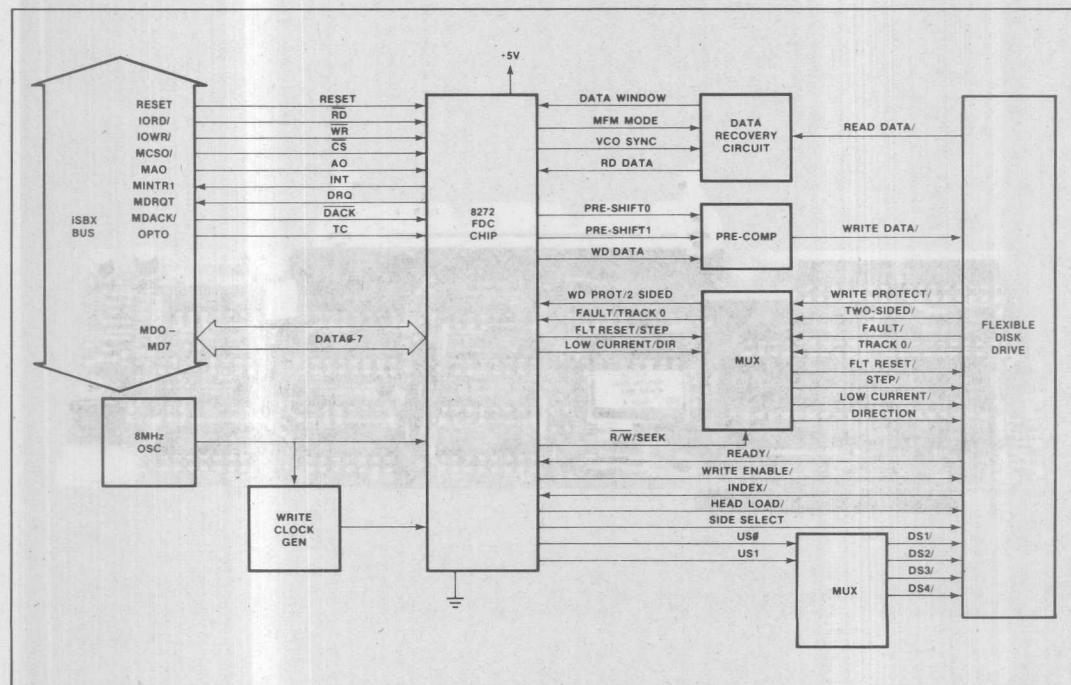


Figure 1. Block Diagram of iSBX 218 Board

PROGRAM INITIATION—All diskette operations are initiated by standard iSBX bus input/output (I/O) operations through the host iSBC single board computer. System software first initializes the controller with the operating characteristics of the selected drive. The diskette is then formatted under program control. Data transfers occur in response to commands output by the CPU.

DATA TRANSFER—Once a diskette transfer operation has been initiated, the controller will

require a data transfer every 13 microseconds (double density) or 26 microseconds (single density). Most CPUs will operate in a polled mode, checking controller status and transferring bytes when the controller is ready. Boards utilizing the Intel 8080 chip, such as the iSBC 80/10B board, will be restricted to single density operation with the iSBX 218 Controller, due to these speed requirements. A programming example illustrating the iSBC 80/10B handler is contained in the Hardware Reference Manual.

SPECIFICATIONS

Compatibility

CPU—Any iSBC single board computer or I/O board compatible with the MULTIBUS system bus and implementing the iSBX bus and connector.

Devices—Double or single density standard (8") and mini (5¼") flexible disk drives. The drives may be single or double sided. Drives known to be compatible are:

Standard (8")		Mini (5¼")	
Caldisk	143M	Shugart	450/400
Remex	RFD 4000	Micropolis	1015-IV
Memorex	550	Pertec	250
MFE	700	Siemens	200-5
Siemens	FDD 200-8	Tandon	TM-100
Shugart	SA 850/800	CDC	9409
Pertec	FD 650	MPI	51/52/91/92
CDC	9406-3		

Diskette—Unformatted IBM Diskette 1 (or equivalent single-sided media); unformatted IBM Diskette 2D (or equivalent double-sided).

Equipment Supplied

iSBX 218 Controller

Reference Schematic

Controller-to-drive cabling and connectors are not supplied with the controller. Cables can be fabricated with flat cable and commercially-available connectors as described in the iSBX 218 Hardware Reference Manual.

Nylon Mounting Bolts

Physical Characteristics

Width—2.85 inches (7.24 cm)

Height—0.5 inches (1.27 cm)

Length—7.5 inches (19.05 cm)

Shipping Weight—1 pound (0.46 Kg)

Mounting—Occupies one double-wide iSBX MULTIMODULE position on boards; increases board height (host plus iSBX board) to 1.13 inches (2.87 cm).

Data Organization and Capacity

Standard Size Drives

	Double Density						Single Density					
	IBM System 34			Non-IBM			IBM System 3740			Non-IBM		
Bytes per Sector	256	512	1024	2048	4096	8192	128	256	512	1024	2048	4096
Sectors per Track	26	15	8	4	2	1	26	15	8	4	2	1
Tracks per Diskette	77			256			77			256		
Bytes per Diskette (Formatted, per diskette surface)	512,512 (256 bytes/sector) 591,360 (512 bytes/sector) 630,784 (1024 bytes/sector)			630,784			256,256 (128 byte/sector) 295,680 (256 bytes/sector) 315,392 (512 bytes/sector)			315,392		

ISBX 218

Drive Characteristics

	Standard Size	Mini Size
	Double/Single Density	Double/Single Density
Transfer Rate (K bytes/sec)	62.5/31.25	31.25/15.63
Disk Speed (RPM)	360	300
Step Rate Time (Programmable)	1 to 16 msec/track in 1 msec increments	2 to 32 msec/track in 2 msec increments
Head Load Time (Programmable)	2 to 256 msec in 2 msec increments	4 to 512 msec in 4 msec increments
Head Unload Time (Programmable)	0 to 240 msec in 16 msec increments	0 to 480 msec in 32 msec increments

Electrical Characteristics

Power Requirements— + 5 VDC @ 0.81A

Environmental Characteristics

Temperature—0°C to 55°C (operating); - 55°C to + 85°C (non-operating).

Humidity—Up to 90% Relative Humidity without condensation (operating); all conditions without condensation or frost (non-operating).

Reference Manual

121583-001—ISBX 218 Flexible Disk Controller Hardware Reference Manual (NOT SUPPLIED).

Reference manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

ORDERING INFORMATION

Part Number **Description**
SBX 218 Flexible Disk Controller

Standard (8")	Mini (5 1/4")
Cadtek 143M	Shugart 450400
Remex FPD 4000	Micrologix 1012-IV
Memorex 550	Petec 550
ITE 100	Siemens 500-5
Siemens FPD 500-5	Tandon TM-100
Shugart SA 550/500	CDC 5400
Petec FPD 550	MPI 51852/192
CDC 9400-3	

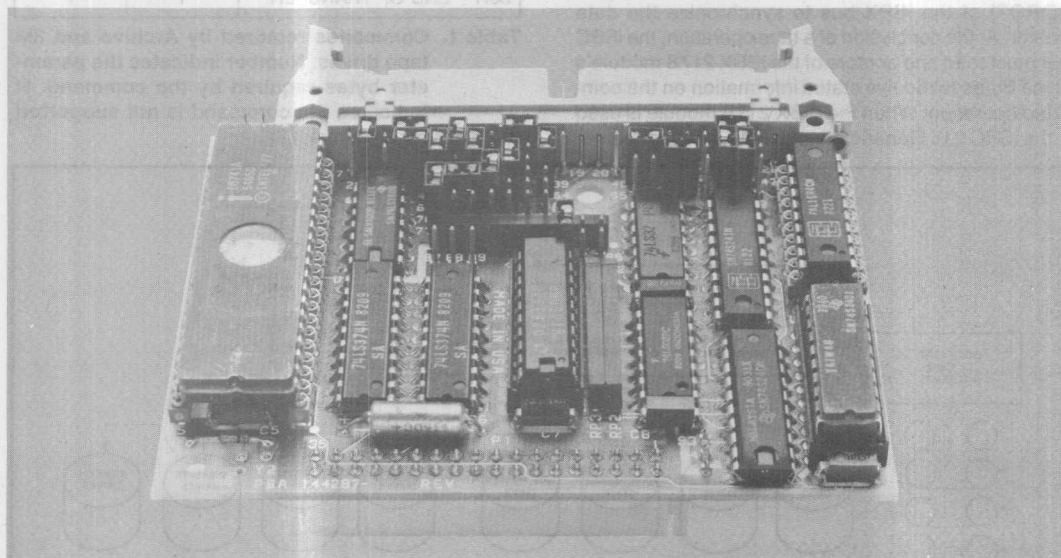
Standard Size Drives		Double Density		Single Density	
		Non-IBM	IBM System 34	Non-IBM	Non-IBM
Bytes per Sector	528	528	528	528	528
Sectors per Track	26	16	8	16	8
Tracks per Diskette	77	77	77	77	77
Bytes per Diskette (Formatted per diskette surface)	812,512	812,512	812,512	812,512	812,512
	(528 bytes/sector)	(528 bytes/sector)	(528 bytes/sector)	(528 bytes/sector)	(528 bytes/sector)
	801,360	801,360	801,360	801,360	801,360
	(512 bytes/sector)	(512 bytes/sector)	(512 bytes/sector)	(512 bytes/sector)	(512 bytes/sector)
	830,784	830,784	830,784	830,784	830,784
	(1024 bytes/sector)	(1024 bytes/sector)	(1024 bytes/sector)	(1024 bytes/sector)	(1024 bytes/sector)



iSBX™ 217B ¼-INCH TAPE DRIVE INTERFACE MULTIMODULE™ BOARD

- iSBX™ MULTIMODULE™ interface provides tape backup capability for iSBC® 215 Generic Winchester Controller
- Configurable to interface with up to four Archive Sidewinder* compatible or four 3M HCD-75 compatible tape drives
- + 5 volt only operation
- Supports transfer rates of 90K, 30K or 17K bytes per second depending on tape speed
- Supported by iRMX™ 88/86 and XENIX† Operating Systems when used on iSBC® 215 Generic Winchester Controller board
- Implements the proposed QIC-2 streaming tape interface standard

The iSBX™ 217B ¼-Inch Tape Drive Interface module is a member of Intel's family of iSBX bus compatible MULTIMODULE™ products. iSBX MULTIMODULE boards plug directly onto any iSBX bus compatible host board, offering incremental on-board I/O expansion. The module is particularly useful for implementing cartridge tape backup capability directly on the iSBC® 215 Generic Winchester Disk Controller via DMA. The iSBX 217B board can also provide a low-cost tape storage interface for any Intel single board computer, with an iSBX connector, via programmed I/O. The iSBX 217 module interfaces with up to four Archive or Cipher Corporation streaming tape drives. These drives provide 20 or 45 megabytes of storage each. When used in conjunction with these drives and the iSBC 215 board, the module can transfer 20 megabytes of data from disk to tape in about fourteen minutes. Alternatively, the iSBX 217B board can interface with up to four 3M Company HCD-75 compatible start/stop tape drives, for those applications requiring access to individual data files on tape.



* Sidewinder is a trademark of Archive Corporation. † XENIX is a trademark of Microsoft Corporation.

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

FUNCTIONAL DESCRIPTION

The iSBX 217B module implements an interface between a host iSBC board and a cartridge 1/4-inch magnetic tape drive, with a minimum of host software overhead. Data transfers may occur in either a direct memory access (DMA) or programmed I/O mode. The DMA mode is available only with host iSBC boards which have DMA capability. In both modes, the host must be able to transfer data at a rate of 90K, 30K or 17K bytes per second, depending on the speed of the tape drive.

Communication with the iSBC® Host

A command plus one-to-five parameter bytes are issued by the host iSBC board to the iSBX 217B module to initiate any tape interface operation. Commands for the Archive and 3M interfaces are summarized in Table 1. If the function is a Read or a Write operation, the host must then be ready to transfer data a byte at a time to or from the module. In programmed I/O mode, with Archive drives, the host polls the iSBX 217B status port to learn when the tape interface is ready for the next 512 byte data block. During the data block transfer, the host is interrupted by MWAIT/ when the interface is ready to transfer a data byte. With 3M tape drives, the host may be interrupted or use MDRQT to detect when the module is ready for the next byte transfer. In DMA mode, the host board uses the DMA Request signal (MDRQT) of the iSBX bus to synchronize the data transfer. At the conclusion of a tape operation, the iSBC host must read one or more of the iSBX 217B module's Sense Bytes to receive status information on the completed operation. When the iSBX 217B module is used on the iSBC 215 Generic Winchester Controller board,

these host requirements are fulfilled by the standard on-board firmware and are transparent to the user.

Command (Hex)	Archive Interface	3M Interface
00H RESET	1	1
01H INITIALIZE	1	1
02H WRITE	1	3
03H WRITE FILE MARK	1	1
04H READ	1	3
05H READ FILE MARK	1	N
06H READ STATUS	1	1
07H REWIND	1	1
08H RETENTION	1	1
09H ERASE TAPE	1	N
0AH RESERVED	N	N
0BH RESERVED	N	N
0CH UNLOAD TAPE	N	1
0DH RESERVED	N	N
—	—	—
13H RESERVED	N	N
14H CONTINUE	N	1
15H WRITE RAM	N	5
16H READ MEMORY	N	5
17H VERIFY	N	5
18H RESERVED	—	—
—	—	—
3FH RESERVED	—	—
40H START OF TRANSFER	1	1
41H RESERVED	—	—
—	—	—
7FH RESERVED	—	—
80H END OF TRANSFER	1	1

Table 1. Commands required by Archive and 3M tape drives. Number indicates the parameter bytes required by the command. N indicates the command is not supported by the drive.

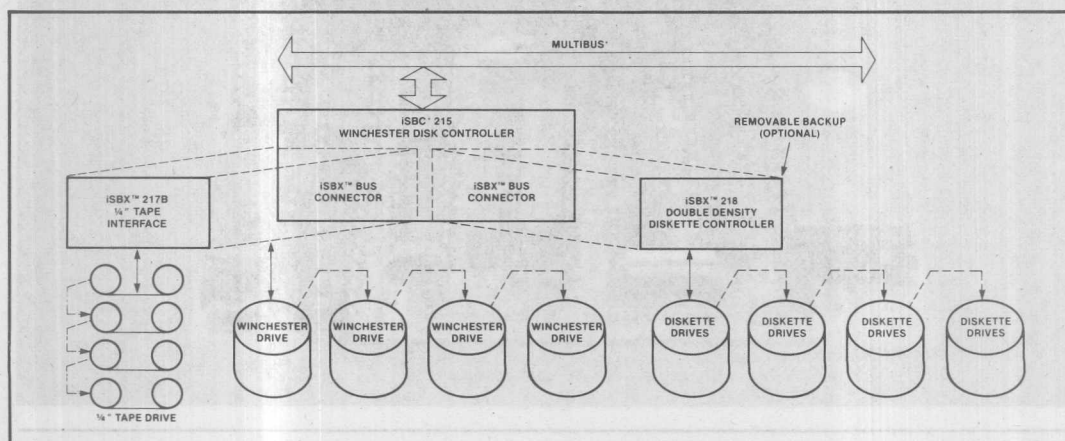


Figure 1. Subsystem Configuration (with Optional Diskette and 1/4" Tape Backup)

SPECIFICATIONS

Compatibility

Host — Any iSBC single board computer or peripheral controller with an iSBX connector. The iSBC 215 Generic Winchester Controller includes on-board firmware to support the iSBX 217B under either the iRMX 88/86 or XENIX Operating Systems. The firmware on the iSBC 215A and iSBX 215B Winchester Controllers cannot support the iSBX 217B module.

Drives — Any Archive Sidewinder or 3M HCD-75 interface compatible cartridge ¼-inch magnetic tape drive.

Transfer Rate

90K (one byte every 11 microseconds), 30K (one byte every 33 microseconds) or 17K (one byte every 53 microseconds) depending on tape drive speed.

Equipment Supplied

iSBX 217B Interface Module
Reference Schematic

Controller-to-drive cabling and connectors are not supplied. Cables can be fabricated with flat cable and commercially-available connectors as described in the iSBX 217B Hardware Reference Manual.

Nylon mounting bolts

Physical Characteristics

Width — 3.08 inches (7.82 cm)

Height — 0.809 inches (2.05 cm)

Length — 3.70 inches (9.40 cm)

Shipping Weight — 3.5 ounces (99.2 gm)

Mounting — Occupies one single-wide iSBC MULTIMODULE position on boards

Electrical Characteristics

Power Requirements — +5 VDC @ 1.5 A

Environmental Characteristics

Temperature — 0°C to 55°C (operating); -55°C to +85°C (non-operating)

Humidity — Up to 90% relative humidity without condensation (operating); all conditions without condensation or frost (non-operating)

Reference Manual

144260-001 — iSBX 217B Board Hardware Reference Manual (NOT SUPPLIED)

ORDERING INFORMATION

Part Number	Description
SBX 217B	Cartridge ¼-Inch Tape Drive Interface



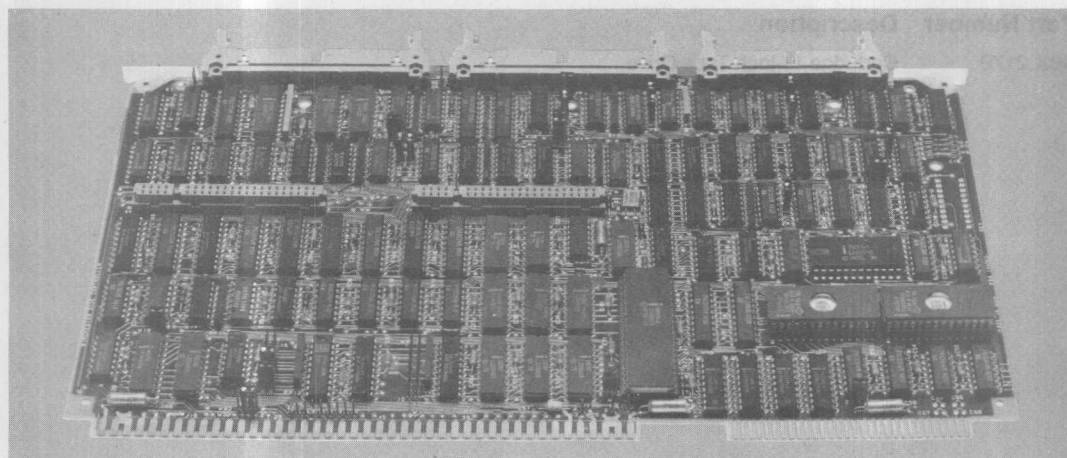
iSBC® 215 GENERIC WINCHESTER CONTROLLER

- Controls up to four 5¼", 8" or 14" Winchester disk drives from over ten different vendors
- Compatible with Industry Standard MULTIBUS® (IEEE 796) Interface
- Supports ANSI X3T9/1226 standard interface
- Software drivers available for iRMX™ 86, iRMX™ 88 and Xenix* Operating Systems
- Intel 8089 I/O Processor provides intelligent DMA capability
- On-board diagnostics and ECC
- Full sector buffering on-board
- Capable of directly addressing 16 MB of system memory
- Removable back-up storage available through the iSBX™ 218 Flexible Disk Controller and the iSBX™ 217 ¼" Tape Interface Module†

Using VLSI technology, the iSBC 215 Generic Winchester Controller (GWC) combines three popular Winchester controllers onto one MULTIBUS board: the iSBC 215A open loop controller, the iSBC 215B closed loop controller, and an ANSI X3T9/1226 standard interface controller. The combined functionality of the NEW iSBC 215 Generic Controller supports up to four 5¼", 8" or 14" Winchester drives from over 10 different drive vendors. Integrated back-up is available via two iSBX MULTIMODULE boards; the iSBX 218 module for floppy disk drives and the iSBX 217 module for ¼" tape units.†

From the MULTIBUS side, the iSBC 215 GWC appears as one standard software interface, regardless of the drive type used. In short, the iSBC 215 GWC allows its user to change drive types without rewriting software. The iSBC 215 Generic Controller is totally downward compatible with its predecessors, the iSBC 215A and 215B controller; allowing existing iSBC 215A and 215B users to move quickly to the more powerful iSBC 215 Generic Winchester Controller. In addition, the new iSBC 215 GWC directly addresses up to 16 megabytes of system memory.

The iSBC 215 controller is used in all of Intel's system products, including the popular SYSTEM 86/330. In addition to its extreme flexibility, the iSBC 215 controller design has withstood well over 15,000 hours of intense system level testing.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. * XENIX is a trademark of Microsoft Corporation. † iSBC® 217 ¼" tape module available late Q4 '82

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Disk Interface

The iSBC 215 Generic Winchester Controller can interface to over 10 different disk drives. To change drive types the user need only reconfigure a minimal number of board jumpers and, if required, insert the proper formatting information into the command parameter blocks.

The ANSI X3T9/1226 standard interface is a simple one-for-one flat cable connection from drive to controller.

Full On-Board Buffer

The iSBC 215 controller contains enough on-board RAM for buffering one full data sector. The controller is designed to make use of this buffer in all transfers. The on-board sector buffer prevents data overrun errors and allows the iSBC 215 Generic Winchester Controller to occupy any priority slot on the MULTIBUS.

ECC

High data integrity is provided by on-board Error Checking Code (ECC) logic. When writing sector ID or data fields, a 32-bit fire code, for burst error correction, is appended to the field by the controller. During a read operation, the same logic regenerates the ECC polynomial and compares this second polynomial to the appended ECC. The ECC logic can detect an erroneous data burst up to 32 bits in length and using an 8089 algorithm can correct an erroneous burst up to 11 bits in length.

iSBX™ Interface

Two iSBX bus connectors provide I/O expansion capability for the iSBC 215 GWC. With the optional addition of the iSBX 218 Flexible Disk Controller MULTIMODULE™ and or the iSBX 217 ¼" Tape Interface Module, the iSBC 215 GWC can be configured into one of four types of peripheral subsystems, see Table 1.

Table 1. Peripheral Subsystem Configurations

	iSBC® 215	iSBX™ 218	iSBX™ 217'
Winchester Only	✓		
Winchester + Floppy	✓	✓	
Winchester + ¼" Tape	✓		✓
Winchester + Floppy + ¼" Tape	✓	✓	✓

NOTE:

1. Available late Q4 '82.

Expanded I/O Capability

The iSBC 215 controller allows the execution of user-written 8089 programs located in on-board or MULTIBUS system RAM. Thus the full capability of the 8089 I/O processor can be utilized for custom I/O requirements.

MULTIBUS® Interface

The iSBC 215 Generic Controller interfaces to the system CPU(s) through MULTIBUS memory. The

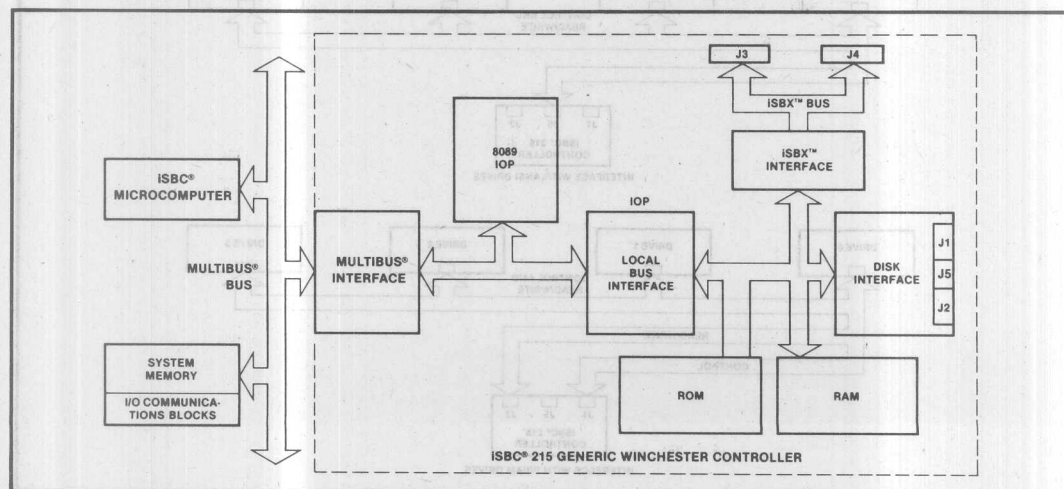


Figure 1. Block Diagram of iSBC® 215 Generic Winchester Disk Controller

iSBC 215 controller directly addresses 16 megabytes of system memory. Commands are passed to and from the iSBC 215 via memory based parameter blocks; these parameter blocks are executed

directly by the iSBC 215 GWC thus off-loading the system CPU(s). Data transfers to and from the iSBC 215 GWC are done via the high speed DMA capability of the Intel 8089 I/O processor.

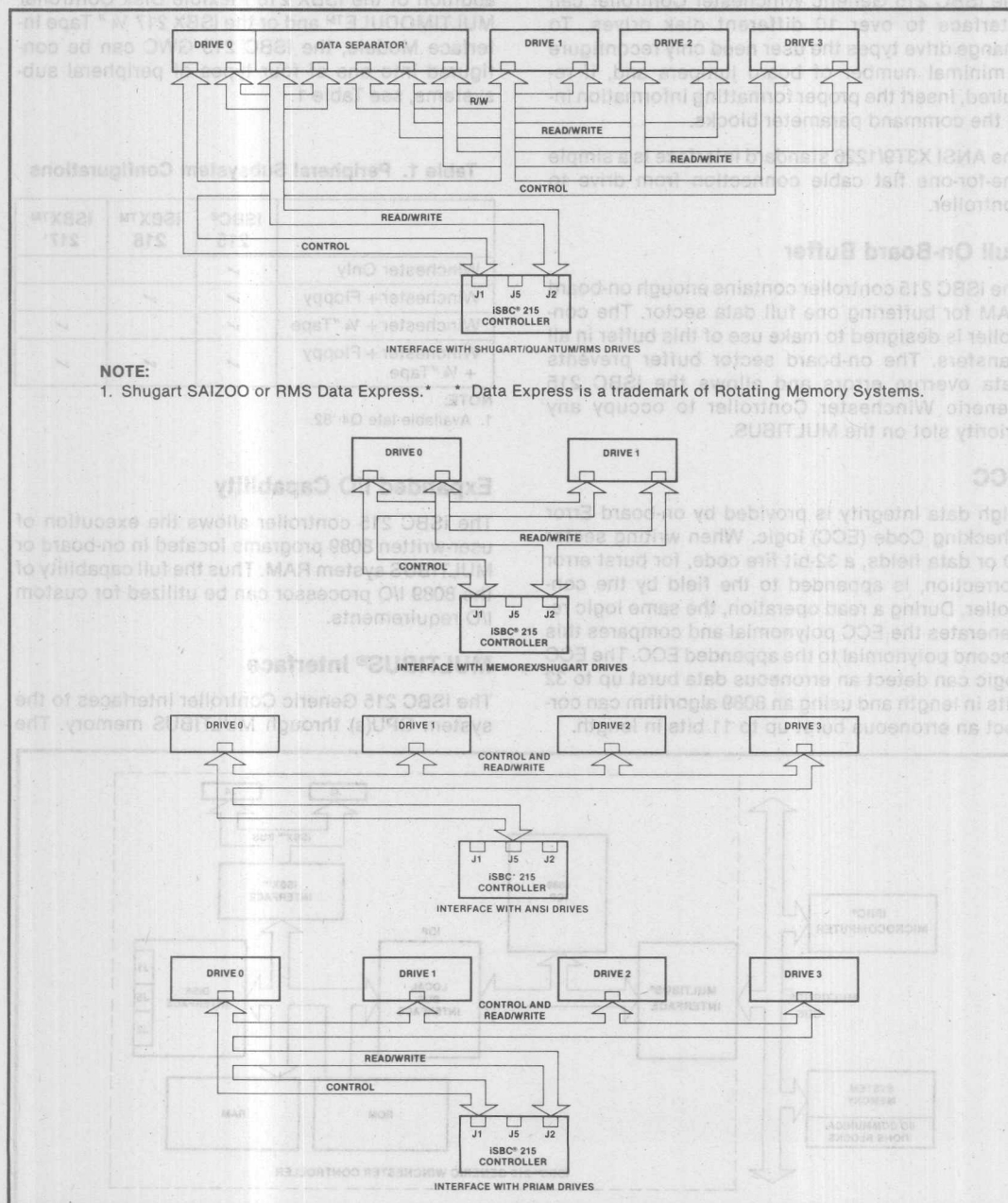


Figure 2. Controller to Drive Interfacing

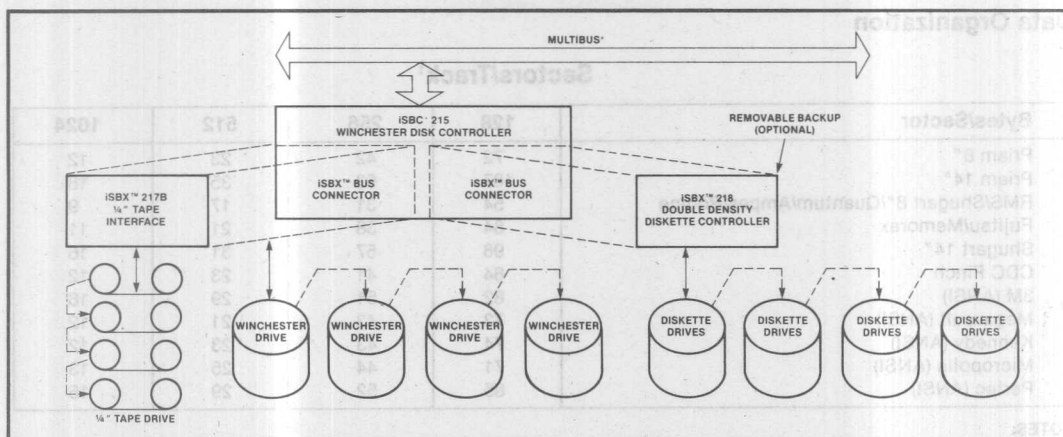


Figure 3. Subsystem Configuration (with Optional Diskette Backup)

SPECIFICATIONS

Compatibility

CPU — Any iSBC MULTIBUS computer or system mainframe.

Disk Drives — Winchester Disk Drives; both open-loop and closed-loop head positioner types. The following drives are known to be compatible:

Open-Loop
Shugart SA 1000 Series Shugart SA 4000 Series Memorex 100 Series Quantum Q2000 Series Fujitsu 2301, 2302 CDC 9410 RMS 5 1/4" Series Rodine 5 1/4" Series Amplex 5 1/4" Series
Closed-Loop
Priam 8" and 14" Drive Series
ANSI
3M 8430 Series Kennedy 6170 Series Micropolis 8" Series Pertec Trackstar Series Priam 8" Series Megavault (SLI) 8" Series
ISBX™ MULTIMODULE™ Boards
ISBX™ 218 Flexible Disk Controller ISBX™ 217 1/4" Tape Interface

Equipment Supplied

iSBC Generic Winchester Controller
Reference Schematic

Controller-to-drive cabling and connectors are not supplied with the controller. Cables can be fabricated with flat cable and commercially-available connectors as described in the iSBC 215 Hardware Reference Manual.

Physical Characteristics

Width — 6.75 in. (17.15 cm)

Height — 0.5 in. (1.27 cm)

Length — 12.0 in. (30.48 cm)

Shipping Weight — 19 oz. (54 kg)

Mounting — Occupies one slot of iSBC system chassis or cardcage/backplane

With an ISBX MULTIMODULE board mounted, vertical height increases to 1.13 in. (2.87 cm).

Electrical Characteristics

Power Requirements

+ 5 VDC @ 3.35A max

+ 5 VDC @ 0.15A max¹

+ 12 VDC @ 0.15A max²

– 12 VDC @ 0.03A max²

Notes:

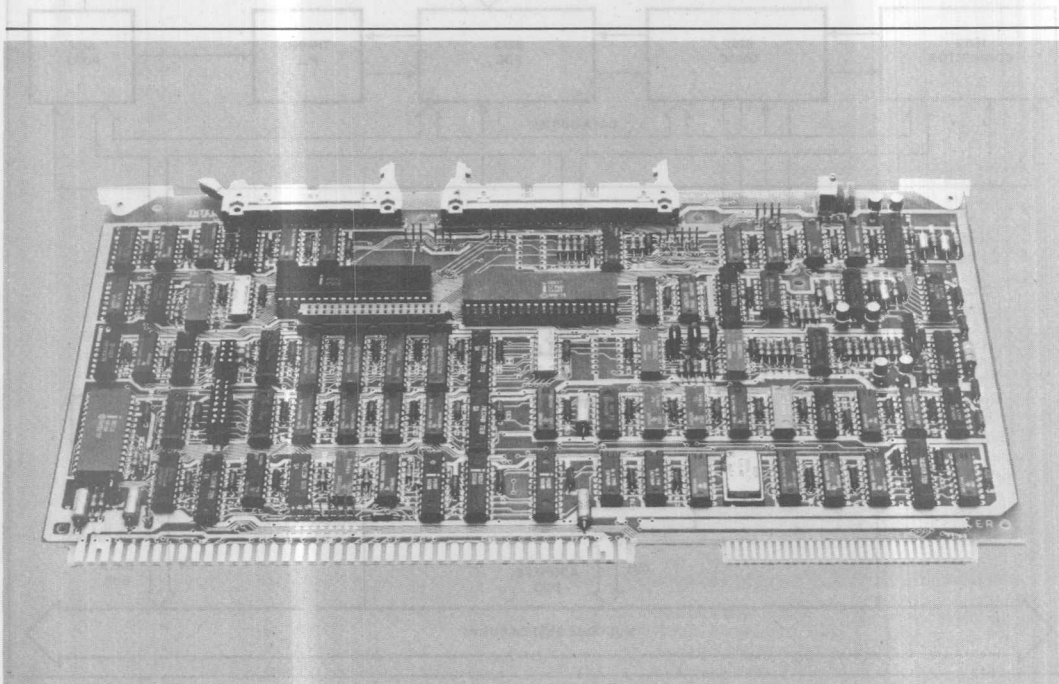
- On-board regulator and jumper allows + 12 VDC usage from MULTIBUS.
- Required for some ISBX MULTIMODULE boards.



iSBC 208 FLEXIBLE DISK CONTROLLER

- Compatible with all iSBC 80, iSBC 86, and iSBC 88 Single Board Computers
- Controls most single and double density diskette drives
- On-board iSBX bus for additional functions
- User-programmable drive parameters allow wide choice of drives
- Phase lock loop data separator assures maximum data integrity
- Read and write on single or multiple sectors
- Single +5V Supply
- Capable of addressing 16M bytes of system memory

The Intel iSBC 208 Flexible Disk Controller is a diskette controller capable of supporting virtually any soft-sectored, double density or single density diskette drive. The standard controller can control up to four drives with up to eight surfaces. In addition to the standard IBM 3740 formats and IBM System 34 formats, the controller supports sector lengths of up to 8192 bytes. The iSBC 208 board's wide range of drive compatibility is achieved without compromising performance. The operating characteristics are specified under user program control. The controller can read, write, verify, and search either single or multiple sectors. Additional capability such as parallel or serial I/O or special math functions can be placed on the iSBC 208 board by utilizing the iSBX bus connection.



FUNCTIONAL DESCRIPTION

Intel's 8272 Floppy Disk Controller (FDC) circuit is the heart of the iSBC 208 Controller. On-board data separation logic performs standard MFM (double density) and FM (single density) encoding and decoding, eliminating the need for external separation circuitry at the drive. Data transfers between the controller and memory are managed by a DMA device which completely controls transfers over the MULTIBUS system bus. A block diagram of the iSBC 208 Controller is shown in Figure 1.

Universal Drives and the iSBC 208 Controller

Because the iSBC 208 Controller has universal drive compatibility, it can be used to control virtually any standard- or mini-sized diskette drive. Moreover, the iSBC 208 Controller fully supports the iSBX bus and can be used with any iSBX module compatible with this bus. Because the iSBC 208 Controller is programmable, its performance is not compromised by its universal drive compatibility. The track-to-track access, head-load, and head-unload characteristics of the selected drive model are program specified. Data may be organized in sectors up to 8192 bytes in length.

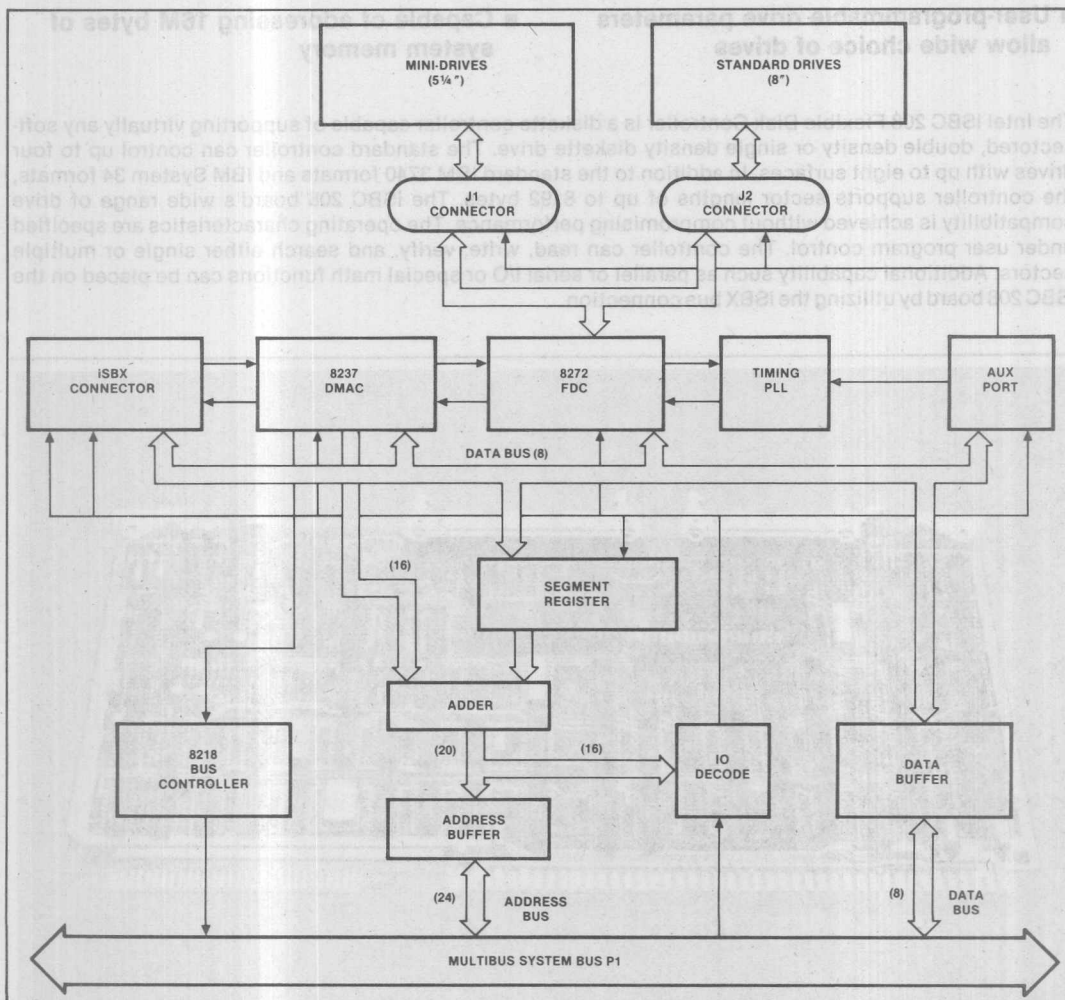


Figure 1. iSBC 208 Flexible Disk Controller Block Diagram

Interface Characteristics

The standard ISBC 208 Controller includes an Intel 8272 Floppy Disk Controller chip which supports up to four drives, single or double sided.

SIMPLIFIED INTERFACE—The cables between the ISBC 208 Controller and the drive(s) may be low cost, flat ribbon cable with mass termination connectors. The mechanical interface to the board is a right-angle header with locking tabs for security of connection.

PROGRAMMING — The powerful 8272 FDC circuit is capable of executing high-level commands that simplify system software development. The device can read and write both single and multiple sectors. CRC characters are generated and checked automatically. Recording density is selected at each Read and Write to support the industry standard technique of recording basic media information on Track 0 of Side 0 in single density, and then switching to double density (if necessary) for operations on other tracks.

Program Initiation—All diskette operations are initiated by standard input/output (I/O) port operations through an ISBC single board computer.

System software first initializes the controller with the operating characteristics of the selected drive. The diskette is then formatted under program control. For subsequent transfers, the starting memory address and transfer mode are specified for the DMA controller. Data transfers occur in response to commands output by the CPU.

Data Transfer—Once a diskette transfer operation has been initiated, the controller acts as a bus master and transfers data over the MULTIBUS at high speed. No CPU intervention is required until the transfer is complete as indicated either by the generation of an interrupt on the bus or by examination of a "done" bit by the CPU.

ISBX BUS SUPPORT — One connector is available on the ISBC 208 board which supports the ISBX system bus. This connector supports single-byte transfer as well as higher-speed transfers supervised by the DMA controller. Transfers may take place in polled or interrupt modes, user-selected. The presence of the ISBX bus allows many different functions to be added to the board. Serial I/O, parallel I/O and various special-purpose math functions are only a few of the capabilities available on ISBX MULTIMODULE boards.

SPECIFICATIONS

Compatibility

CPU—Any ISBC MULTIBUS computer or system main frame

Devices—Double or single density standard (8") and mini (5¼") flexible disk drives. The drives may be single or double sided. Drives known to be compatible are:

Standard (8")		Mini (5¼")	
Caldisk	143M	Shugart	450 SA 400
Remex	RFD 4000	Micropolis	1015-IV
Memorex	550	Pertec	250
MFE	700	Siemens	200-5
Siemens	FDD 200-8	Tandon	TM-100
Shugart	SA 850/800	CDC	9409
Pertec	FD 650	MPI	51/52/91/92
CDC	9406-3		

Diskette—Unformatted IBM Diskette 1 (or equivalent single-sided media); unformatted IBM Diskette 2D (or equivalent double-sided).

Equipment Supplied

ISBC 208 Controller
Reference Schematic
Controller-to-drive cabling and connectors are not supplied with the controller. Cables can be fabricated with flat cable and commercially-available connectors as described in the ISBC 208 Hardware Reference Manual

Physical Characteristics

Width—6.75 inches (17.15 cm)
Height—0.5 inches (1.27 cm)
Length—12.0 inches (30.48 cm)
Shipping Weight—1.75 pounds (0.80 Kg)
Mounting—Occupies one slot of ISBC system chassis or ISBC 604/614 Cardcage/Backplane. With an ISBX MULTIMODULE board mounted, vertical height increases to 1.13 inches (2.87 cm).

Electrical Characteristics

Power Requirements— + 5 VDC @ 3.0A

ISBC 208

Data Organization and Capacity

Standard Size Drives

	Double Density						Single Density					
	IBM System 34			Non-IBM			IBM System 3740			Non-IBM		
Bytes per Sector	256	512	1024	2048	4096	8192	128	256	512	1024	2048	4096
Sectors per Track	26	15	8	4	2	1	26	15	8	4	2	1
Tracks per Diskette	77			256			77			256		
Bytes per Diskette (Formatted, per diskette surface)	512,512 (256 bytes/sector) 591,360 (512 bytes/sector) 630,784 (1024 bytes/sector)			630,784			256,256 (128 byte/sector) 295,680 (256 bytes/sector) 315,392 (512 bytes/sector)			315,392		

Drive Characteristics

	Standard Size	Mini Size
	Double/Single Density	Double/Single Density
Transfer Rate (K bytes/sec)	62.5/31.25	31.25/15.63
Disk Speed (RPM)	360	300
Step Rate Time (Programmable)	1 to 16 msec/track in 1 msec increments	2 to 32 msec/track in 2 msec increments
Head Load Time (Programmable)	2 to 254 msec in 2 msec increments	4 to 508 msec in 4 msec increments
Head Unload Time (Programmable)	16 to 240 msec in 16 msec increments	32 to 480 msec in 32 msec increments

Environmental Characteristics

Temperature—0°C to 55°C (operating); -55°C to +85°C (non-operating)

Humidity—Up to 90% Relative Humidity without condensation (operating); all conditions without condensation or frost (non-operating)

Reference Manual

143078-001—iSBC 208 Flexible Disk Controller Hardware Reference Manual (NOT SUPPLIED). Reference manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

ORDERING INFORMATION

Part Number	Description
SBC 208	Flexible Disk Controller

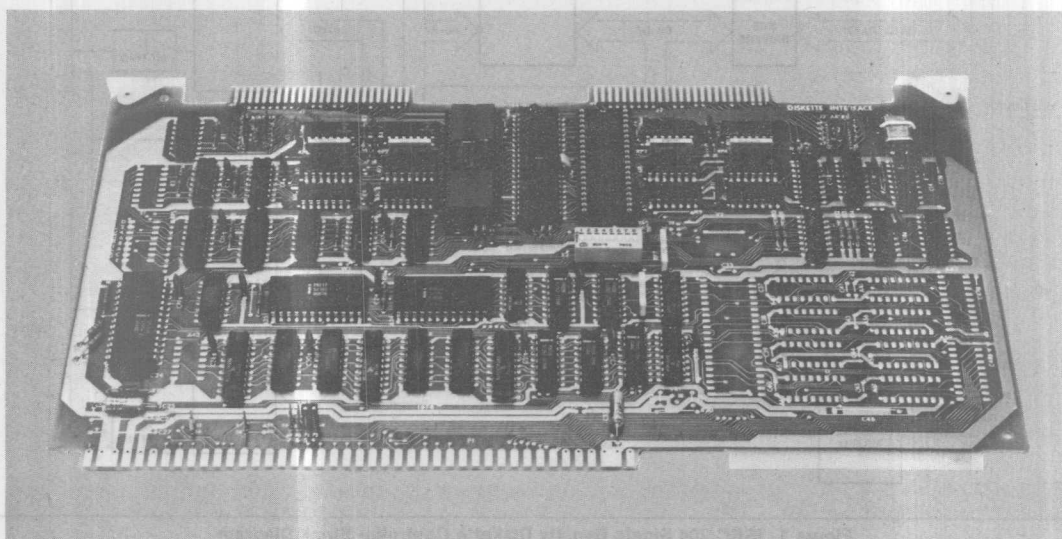
Mini (5 1/4")	Standard (5 1/4")
Shugart 450 SA 400	Caldisk 450
Micrologix 4015 IV	Remex RFD 4000
Perlec 500	Memorex 500
Siemens 500-8	MFE 700
Tandon TM-100	Siemens RFD 300-8
QDC 8408	Shugart SA 500/800
MPI 5125/8192	Perlec FD 500
	CDC 8408-3



iSBC 204 SINGLE DENSITY FLEXIBLE DISKETTE CONTROLLER

- Full compatibility with iSBC 80, iSBC 86, and iSBC 88 Single Board Computers
- Direct compatibility with most single-density, soft-sectored standard- (8") and mini-size (5 1/4") flexible diskette drives
- Software supported by iRMX 80, iRMX 86 and iRMX 88 Real-Time Multi-tasking Executive disk file system
- Support by CP/M operating system
- DMA input/output allows single board computers to process in parallel with diskette transfer operations
- Programmable track-to-track access, head-settling, and head-load times
- On-board data separation logic
- Read, write, verify, and search on single or multiple sectors
- Single +5V supply

The Intel iSBC 204 Single Density Flexible Diskette Controller is a single board universal diskette controller capable of supporting virtually any software-sectored, single density diskette drive. The standard iSBC 204 Controller can control two drive surfaces (two single-sided drives or one double-sided drive). With the addition of a second (optional) Intel 8271 component, up to four drives can be supported. In addition to the standard IBM 3740 formats, the controller supports sector lengths of up to 4096 bytes plus mini-size drive formats. The iSBC 204's wide range of drive compatibility is achieved without compromising performance. The operating characteristics (track-to-track access, head-load, and head-settling times) are specified under user program control. The controller can read, write, verify, and search either single or multiple sectors.



FUNCTIONAL DESCRIPTION

Intel's 8271 Floppy Disk Controller (FDC) circuit is the heart of the iSBC 204 Controller. On-board data separation logic performs standard FM encoding and decoding, obviating external separation circuitry at the drive. Diskette data transfers are DMA (direct memory access) through an on-board Intel 8257 DMA controller circuit which manages DMA transfers and signals the master iSBC processor on completion of the transfer. A block diagram of the iSBC 204 Controller is shown in Figure 1.

Universal Drive and MULTIBUS Compatibility

Because the iSBC 204 Controller has universal drive compatibility, it can be used to control virtually any standard- or mini-sized single density diskette drive. Moreover, the iSBC 204 Controller fully supports the microcomputer industry standard MULTIBUS system bus and can be used with any single board computer or system compatible with Intel's bus. Because the iSBC 204 Controller is programmable, its performance is not compromised by its universal drive compatibility. The track-to-track access, head-load, and head-settling characteristics of the selected drive model are program specified. Data may be organized in a fully compatible IBM 3740 sector format, in sectors up to 4096 bytes in length, or in formats compatible with the mini-sized diskette drives.

Interface Characteristics

Expandability — Each standard iSBC 204 Controller includes a single 8271 FDC circuit capable of supporting two drive surfaces. Optionally the iSBC 204 may be expanded to support four single-sided (or two double-sided) drives with the insertion of a second 8271 component into an on-board socket.

Simplified Interface — The cables between the iSBC 204 Controller and the drive(s) may be either low cost, flat ribbon cable with mass termination connectors or twisted pair conductors with individually wired connectors. An on-board, cross-connect matrix allows optional drive control and status signals to be connected while maintaining pin-to-pin compatibility.

Programming

The powerful 8271 FDC circuit is capable of executing high-level commands that simplify system software development. The device can read, write, and verify both single and multiple sectors. CRC characters are generated and checked automatically. Up to two tracks on each surface may be designated "bad" and logically removed from the diskette.

Sector Scanning — Scan commands permit sectors to be searched for a specified data pattern or "key". During scan operations the pattern image from memory is continuously compared with a sector or multiple sectors

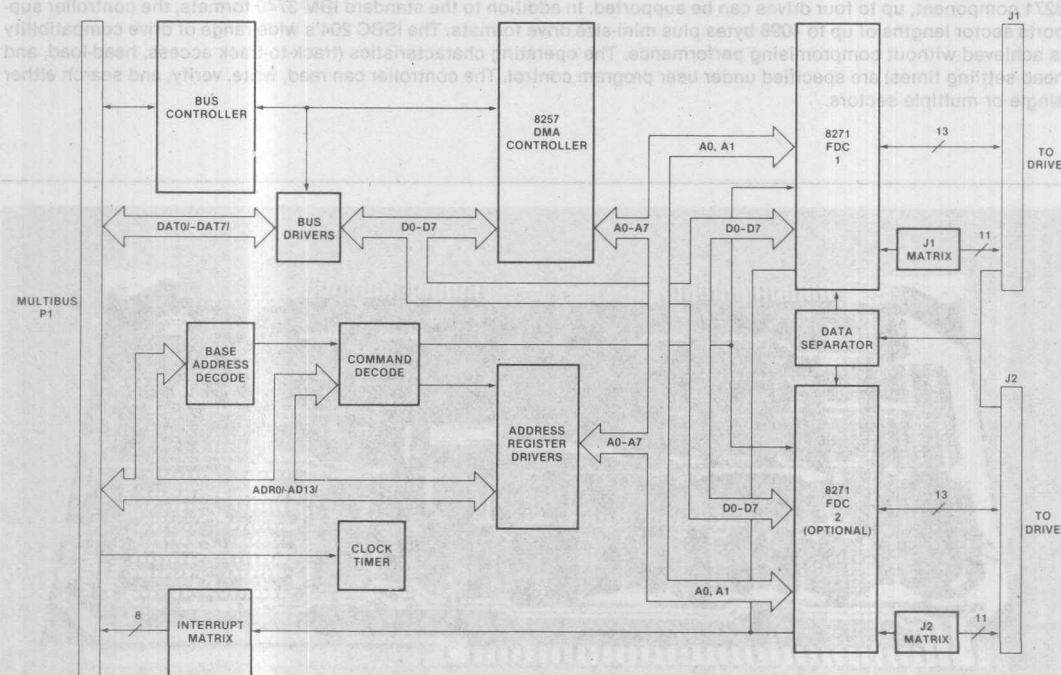


Figure 1. iSBC 204 Single Density Diskette Controller Block Diagram

ISBC 204

read from the diskette. No CPU intervention is required until a match is found or all specified sectors have been searched.

Program Initiation — All diskette operations are initiated by standard input/output (I/O) port operations through an iSBC single board computer. System software first initializes the controller with the operating characteristics of the selected drive. The diskette is then formatted under program control. For subsequent transfers, the starting memory address and transfer

mode are specified for the DMA controller. Data transfers occur in response to commands output by the CPU.

Data Transfer — Once a diskette transfer operation has been initiated, the controller acts as a bus master and transfers data over the MULTIBUS at high speed. No CPU intervention is required until the transfer is complete as indicated either by the generation of an interrupt on the bus or by examination of a "done" bit by the CPU.

SPECIFICATIONS

Compatibility

CPU — Any iSBC MULTIBUS computer or system mainframe.

Drive — Single density, standard- (8") and mini-sized (5 1/4 ") diskette drives. The standard iSBC 204 Controller supports two single-sided drives or one double-sided drive. By adding an (optional) 8271 FDC, four single-sided or two double-sided drives may be supported. The following drives are known to be compatible:

Standard Size	Mini Size
CDC 9404 GSI 110 MEMOREX 550 MEMOREX 552 (dual-sided) SHUGART 800 SHUGART 850 (dual-sided) WANGCO 76S PERTEC 650 (SD/DD, DBL. Head)	PERTEC FD200 SHUGART SA400 WANGCO 82

Diskette — Unformatted IBM Diskette 1 (or equivalent single-sided); unformatted IBM Diskette 2 (or equivalent double-sided); unformatted Shugart SA104 Diskette (or equivalent mini).

Data Organization and Capacity (Standard Size Drives)

	IBM Format			Non-IBM Format		
Bytes per sector	128	256	512	1024	2048	4096
Sectors per track	26	15	8	4	2	1
Tracks per diskette	77			Up to 255		
Bytes per diskette (77 tracks)	256,256 (128-byte sector) 295,680 (256-byte sector) 315,392 (512-byte sector)			315,392		

Drive Characteristics

	Standard Size	Mini Size
Transfer rate (KB/sec)	250	125
Disk speed (RPM)	360	300
Track-to-track access (programmable)	1 to 255 ms in 1 ms steps	2 to 510 ms in 2 ms steps
Head settling time (programmable)	0 to 255 ms in 1 ms steps	0 to 510 ms in 2 ms steps
Head load time (programmable)	0 to 60 ms in 4 ms steps	0 to 120 ms in 8 ms steps

Equipment Supplied

iSBC 204 Controller

Reference Schematic

Controller-to-drive cabling and connectors are not supplied with the iSBC 204 Controller. Cables can be fabricated easily using either flat ribbon cable or twisted pair conductors with commercially available connectors as described in the iSBC 204 Hardware Reference Manual.

Optional Equipment

8271 Flexible Diskette Controller Component — Adding a second 8271 device to the fully tested circuit on the iSBC 204 Controller allows four drive surfaces to be supported.

Physical Characteristics

Width — 6.75 in. (17.15 cm)

Height — 0.5 in. (1.27 cm)

Length — 12.0 in. (30.48 cm)

Shipping Weight — 1.75 lb (0.80 kg)

Mounting — Occupies one slot of iSBC system chassis or iSBC 604/614 cardcage.

Electrical Characteristics

Power Requirements — 5.0V ($\pm 5\%$), 2.5A max

Environmental Characteristics

Temperature — 0°C to 55°C (operating); -55°C to +85°C (non-operating)

Humidity — Up to 90% relative humidity without condensation (operating); all conditions without condensation or frost (non-operating)

Reference Manuals

9800568 — iSBC 204 Diskette Controller Hardware Reference Manual (NOT SUPPLIED).

9800522 — RMX/80 User's Guide (NOT SUPPLIED).

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

SBC 204	Universal Flexible Diskette Controller
---------	--



APPLICATION NOTE

AP-118

Data Transfer. — Once a diskette transfer operation has been initiated, the controller acts as a bus master and transfers data over the MULTIBUS at high speed. No CPU intervention is required until the transfer is complete as indicated either by the generation of an interrupt on the bus or by examination of a "done" bit by the CPU.

Program Initiation. — All diskette operations are initiated by standard input/output (I/O) port operations through an 18BC single board computer. System software first initializes the controller with the operating characteristics of the selected drive. The diskette is then formatted under program control. For subsequent transfers, the starting memory address and transfer

SPECIFICATIONS

Compatibility

June 1981

18BC Mini-TURBO computer or system using 18BC 204 Controller.

Drive — Single density, standard (5 $\frac{1}{4}$ " and mini-sized 5 $\frac{1}{4}$ " diskette drive. The standard 18BC 204 Controller supports two single-sided drives or one double-sided drive. By adding an optional 8271 FDC, four single-sided or two double-sided drives may be supported. The following drives are known to be compatible:

Standard Size	Mini Size
QDD 9444	PERTEC F2000
QSI 110	SHUGART 2400
MEMOREX 510 (dual-sided)	WANGCO 33
SHUGART 300	
SHUGART 850 (dual-sided)	
WANGCO 750	
PERTEC 800 (HD) 300 (LD)	

Diskette — Unformatted IBM Diskette (single-sided), unformatted IBM Diskette (double-sided), unformatted Shugart 2400 (double-sided), unformatted Shugart 300 (double-sided), unformatted Shugart 850 (double-sided).

Data Organization and Capacity (Standard Size Drives)

Bytes per sector	150	250	510	1020	2040	4080
Sectors per track	26	13	6	3	1	1
Tracks per diskette	5	5	5	5	5	5
Bytes per diskette	204,000	102,000	51,000	25,500	12,750	6,375
150 byte sector (72 tracks)	150	250	510	1020	2040	4080
250 byte sector	150	250	510	1020	2040	4080
510 byte sector	150	250	510	1020	2040	4080
1020 byte sector	150	250	510	1020	2040	4080
2040 byte sector	150	250	510	1020	2040	4080

Drive Characteristics

Transfer rate (bytes/sec)	Standard Size	Mini Size
1000	250	150
2000	250	150
3000	250	150
4000	250	150
5000	250	150
6000	250	150
7000	250	150
8000	250	150
9000	250	150
10000	250	150

**Raster Graphics Techniques
for MULTIBUS® Users**

John Wipfli

INTRODUCTION

Advances in microelectronics and telecommunications are changing the way in which information is moved and managed. Certainly, computer graphics will be called upon to represent much of the technical and business information in the coming years. Through Intel's high performance single board computers and low cost memory technology, highly effective implementations of computer graphics display systems are achievable today.

Computer graphics is one of the many areas of applications which once was considered the domain of the minicomputer and now may be handled by Intel® microcomputers.

The purpose of this note is to acquaint you with some of the hardware and software techniques which are required to form useful display systems with high performance microcomputers and raster graphics display devices. The system discussed in this note will generate displays of area filled polygons as well as simple line drawings, both of which are typical of business and engineering information encountered in the office, laboratory, and factory environments.

This application note will lead the reader through a complete implementation example of the hardware and software of a display system which will produce either a 256 × 240 pixel color display or a 512 × 480 pixel monochromatic (black and white) display. First, the note discusses raster scan display devices and the demands that their parameters place on a microcomputer-based controller. Then, a MULTIBUS®-based frame buffer (bit mapped) prototype display controller is described. After considering the programmer's view of this hardware, the graphics support software is outlined from its PL/M-86 implementation.

Though the frame buffer board described in this note is not an off-the-shelf Intel® product, the straightforward implementation demonstrates how easily a display system capability can be added to the MULTIBUS® system bus for applications which require graphics.

Throughout this note are numerous references to the excellent text on computer graphics by Newman and Sproull¹. If you are new to the field of computer graphics it is recommended that you read the chapters on raster graphics.

RASTER SCAN GRAPHIC DISPLAYS

Of the various alternatives for graphics display devices (direct view storage tube, calligraphic display, plasma panels, etc.) the raster scan cathode ray tube (CRT)

display is a dominant technology. The display electronics has the advantage of volume production economics, driven by the consumer television industry, and the display controller enjoys price reductions offered by the random access memory (RAM) experience of the semiconductor industry.

Many CRT computer terminal displays already offer a limited graphics capability by allowing the user to select, and in some cases, even reprogram, a set of graphics characters which can be used to compose a limited amount of graphic scenes. Intel® will soon offer the iSBX 270 MULTIMODULE Video Display Controller which can be added to any Intel® Single Board Computer with iSBX MULTIMODULE board capability. The iSBX 270 board is based on the Intel® 8275 and is primarily a character display controller although limited graphics capabilities are available by changing the character generator of the board. Such displays can be quite effective for a particular application and because of LSI display controller components like the Intel® 8275 and 8276 the cost of these display systems is quite low.

This application note is directed exclusively at a display system with full graphics capability. Such raster scan graphic display systems are referred to as BIT-MAPPED DISPLAYS, where each PIXEL (picture element) of video information is mapped into unique bit(s) of memory, or as FRAME BUFFER DISPLAYS, because digital memory is used to store a copy of a frame of video information.

Raster Scan CRT Operation

To understand how computer-generated images are formed it is useful to first understand how a home television receiver produces a picture. The display hardware in a television receiver is approximately the same as that used in computer-based raster display systems.

Broadcast television represents each portion of a picture as a FRAME of information. The television frame is the electronic analogy to a frame of movie film. Unlike the analog image on a film frame, a television frame is composed of a discrete number of SCAN LINES. To produce a picture, the CRT's deflection circuits force an electron beam to scan across the face of the CRT from left to right, modulating the beam's intensity in proportion to the brightness of the corresponding line of the picture. The beam excites the inside surface of the phosphorous-coated CRT face and produces light. This scanning operation is repeated in sequence from the top of the screen to the bottom for each of the scan lines which comprise the image. As the electron beam arrives at the right of the screen at the completion of each scan line, the beam is first BLANKED and then DRIVEN to RETRACE a path back to the left of the screen in

preparation for the next scan line. The blanking is performed so that no extraneous image forms as the beam passes from right to left. Similar to the HORIZONTAL SYNCHRONIZATION process just described is that for the vertical axis of the display. To force the beam to start forming the picture with the upper leftmost corner of the screen, a VERTICAL SYNCHRONIZATION signal causes the beam to be swept from the bottom of the CRT to the top. During this interval the beam also must be blanked to avoid corruption of the picture. Thus, there are five major signals which control the display process in a CRT display:

Horizontal Synchronization (or Horizontal Drive)

Horizontal Blanking

Vertical Synchronization (or Vertical Drive)

Vertical Blanking

Blanked Video Intensity

Most raster scan monitors derive their basic specifications from broadcast television standards. Serious video enthusiasts should review the Electronic Industries Association standards in this area. Three such interesting documents are listed in the Bibliography^{2,3,4}. These standards specify the analog voltage combination of video picture information and synchronization control signals which must be produced. This composite video signal is electrically complex because it must be broadcast over a single television channel and be bandwidth limited to a few megahertz. CRT data monitors often abandon the composite video approach and allow the individual constituents of the signal to be applied separately as blanked video data, horizontal synchronization (drive), and vertical synchronization (drive). Also, in the separated video approach, TTL signal levels are allowed at the monitor interface to simplify the controller design.

Video Interfacing

Raster scan display monitors may be purchased with a variety of interface configurations. One type of interface which is widely used is known as COMPOSITE VIDEO. Composite video is so named because information is supplied to the monitor on a single coaxial cable which carries an analog combination of the video, blanking, and synchronization signals described earlier. In a color system, this composite signal becomes complex to produce since it consists of a vector combination of the red, green, and blue video data and the synchronization information. This composite signal is decomposed by the monitor into the constituent elements which are required to form an image.

Another type of interface is the separate sync variety. With monitors of this style, the user provides blanked video and synchronization on separate cables. For a color system, the blanked video is usually supplied on

three separate coaxial cables, carrying red, green, and blue video information. These monitors are often called RGB monitors, signifying the individual control which the separate video cables provide. In a monochrome system, the blanked video is supplied on a single cable. Furthermore, though many monitors allow proportional analog voltage control of beam intensity, TTL signals are usually accepted directly and produce fully saturated beam intensities.

Regardless of the monitor chosen, a set of specifications for synchronization, blanking, and video must be met. Though monitors vary widely in the precise requirements for these signals there are a few common denominators based on commercial television standards.

In countries with power systems based on 60 hertz operation, frames are sequenced to a television set at a rate of 30 times per second, implying that a new field is encountered 60 times per second. (See the description of interlaced fields in the next section on INTERLACING.) In countries using 50 hertz power, these frequencies are scaled appropriately. Throughout this note, a 60 hertz power system will be assumed.

The vertical synchronization signal occurs at a rate of 60 hertz. With two interlaced vertical fields per video frame, the frame rate is 30 hertz. 525 horizontal scan lines per frame implies a horizontal synchronization rate of 15750 hertz $[(30 \text{ frame/second}) \cdot (525 \text{ horizontal scan lines/frame})]$. A horizontal scan line requires 63.49 microseconds $(1/15750 \text{ hertz})$.

The width and positioning of the synchronization signals (with respect to the blanked video data) must be tailored to the particular monitor to best center the information on the CRT face.

Of the 63.49 microsecond period of a horizontal scan line, some 10 to 15 microseconds are normally required to retrace the electron beam to the left of the CRT in preparation for the next scan line. During this retrace period the video data stream must be blanked. Similarly, a vertical retrace period occurs at the start of each new field of the interlaced display during which time the video data must also be blanked. Typically 450 to 480 of the 525 total scan lines of a display are useable for the presentation of information, the rest being consumed by the vertical retrace periods.

Interlacing

CRT monitors used for television display a frame of information as two INTERLACED FIELDS of scan lines. The interlace process simply divides the frame into two sets of scan lines. First, one field (set of scan lines) is displayed by the raster scan technique. Then, the elec-

tron beam is retraced to the top left corner of the CRT, offset by one half of the vertical spacing of scan lines, and the second field is scanned. Thus half of the picture is displayed on one scan and the other half on a subsequent scan. Figure 1 compares the travel of the electron beam on a non-interlaced display with that of an interlaced display.

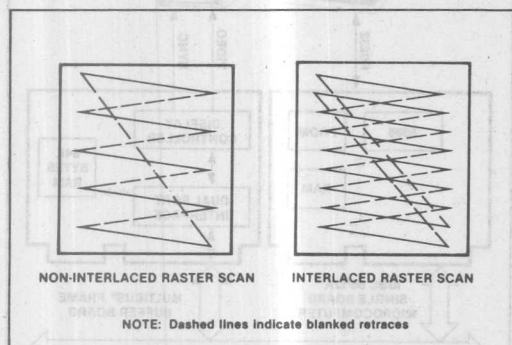


Figure 1. Interlacing

CRT data terminals normally utilize only one of these fields when forming the ubiquitous 1920 (24 rows of 80 columns) character display. There are two reasons for this. First, the display controller electronics is somewhat easier to construct without the interlace requirement. Perhaps more importantly, the problem of display FLICKER is avoided by making the frame rate equal to the field rate. High resolution graphics displays require the use of both fields to achieve maximum resolution. In these graphics displays, flicker can be an annoying problem.

Display Flicker

The phosphorescent coating applied to the inside surface of a CRT exhibits a PERSISTENCE to continue to emit photons after a region has been excited by the electron beam. Most conventional monochrome CRT displays and television monitors utilize a tube with P4 phosphor which decays to 0.1% of its original intensity in 20 milliseconds. The persistence of the P4 phosphor is long enough for the human eye to capture the information in a single frame of a television display and yet not so long that a sequence of fast changing action leaves artifacts or ghosts of the previous frames behind to confuse the viewer. Furthermore, television scenes produced with typical cameras are incapable of representing features which occupy a single scan line of the display. Information on one scan line of a camera-generated image unavoidably carries some of the information of the scan line above it and the scan line below it. Remember that the adjacent scan lines are in different fields. As these camera-generated lines are displayed, the eye blends the information together to form an apparently flicker-free image.

When an interlaced frame of video information is digitally-generated, adjacent pixels may be (and often are) of greatly different intensities. Furthermore, because of interlacing, the adjacent pixels on a horizontal scan line are excited by the electron beam at a rate of 30 times per second, once each frame. Vertically neighboring pixels, produced by members of alternate fields of the frame, are illuminated at intervals of 1/60 of a second. See Figures 2 and 3 which illustrate the differing refresh intervals for adjacent pixels on horizontal and vertical lines. Notice that for the horizontal row of pixels, the beam visits the group once every two fields. However, for a vertical column of pixels the beam illuminates one half of the line with field 0 and the other half of the line with field 1. Half of the vertical column of pixels is drawn during each field. This disparity of refresh rates causes horizontal lines, and lines with primarily horizontal segments, to flicker noticeably if the monitor employs the conventional P4 phosphor used in character displays.

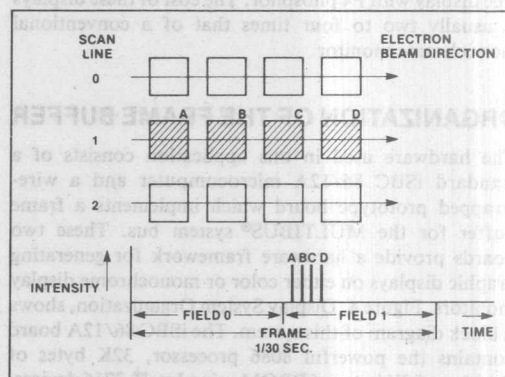


Figure 2. Horizontal Row of Pixels

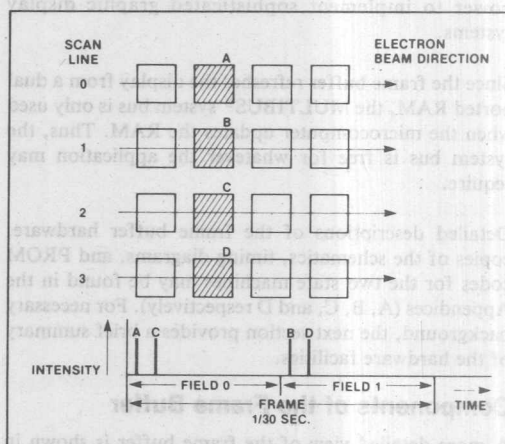


Figure 3. Vertical Column of Pixels

One answer to this problem is quite straightforward and requires that the monitor be equipped with an optional (often at no extra cost) phosphor. EIA P31 or EIA P39 phosphor produces satisfactory flicker-free displays but retains standard monitor interface. For the monochrome display option in this application note, the choice was made to select a long persistence phosphor (P39) and preserve the conventional interlaced display monitor interface.

Another approach to eliminate flicker is to abandon interlacing, discard the information in one field, and thereby make the frame rate equal to the field rate of 60 per second. Unfortunately, this also limits the vertical resolution possible with a conventional monitor to one half of that which is available. The color display option was implemented by using a non-interlaced display format for this application. Higher performance non-interlaced monochrome monitors are now available which can offer 800 to 1024 horizontal lines of flicker-free display with P4 phosphor. The cost of these displays is usually two to four times that of a conventional monochrome monitor.

ORGANIZATION OF THE FRAME BUFFER

The hardware used in this application consists of a standard iSBC 86/12A microcomputer and a wire-wrapped prototype board which implements a frame buffer for the MULTIBUS[®] system bus. These two boards provide a hardware framework for generating graphic displays on either color or monochrome display monitors. Figure 4, Display System Organization, shows a block diagram of this system. The iSBC 86/12A board contains the powerful 8086 processor, 32K bytes of RAM, and 8K bytes of PROM using Intel[®] 2716 devices. This provides sufficient program storage and processing power to implement sophisticated graphic display systems.

Since the frame buffer refreshes the display from a dual ported RAM, the MULTIBUS[®] system bus is only used when the microcomputer updates the RAM. Thus, the system bus is free for whatever the application may require.

Detailed descriptions of the frame buffer hardware, copies of the schematics, timing diagrams, and PROM codes for the two state machines may be found in the Appendices (A, B, C, and D respectively). For necessary background, the next section provides a brief summary of the hardware facilities.

Components of the Frame Buffer

A more detailed view of the frame buffer is shown in Figure 5, Block Diagram of Frame Buffer. The buffer

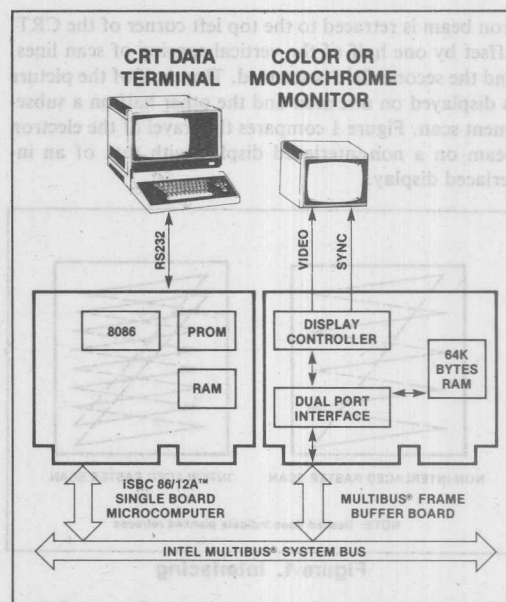


Figure 4. Display System Organization

itself is composed of two RAM arrays named "A" and "B", each containing 32K bytes of dynamic RAM. This RAM may be accessed as 8-bit bytes or 16-bit words. Two ports to the RAM are provided, the first for MULTIBUS[®] access and the other for use by the display controller section. A two port arbitration logic section schedules accesses to the RAM array, guaranteeing that the display controller receives timely access and interleaving MULTIBUS[®] access when the RAM is unused by the display controller. A central timing unit based on a 10 megahertz crystal reference controls activity on the entire board, including the precise signals needed to control the dynamic RAM array.

Two finite state machines, one for control of horizontal synchronization and one for vertical synchronization of the display, adapt the general hardware to specific requirements of different monitors. These two state machines are microprogrammed using 2716 PROMs and may be reprogrammed for added flexibility. The state machines control the advance of the video refresh address counters which scan the RAMs in sequence to obtain all the necessary bytes to form the display. Four shift registers capture RAM data and serialize it for the video output circuits. The shift registers are appropriately named RED, GREEN, BLUE, and XTRA.

Video output circuits combine the shift register data with blanking and synchronization information from the state machines to produce the video signals needed to drive a variety of display monitors.

The frame buffer may be used to control either a color display or a monochromatic display. A hardware jumper selects one of the display modes.

Color Mode

This frame buffer can be employed in color mode or in monochrome mode. In color mode, three blanked video data outputs are provided (RED, GREEN, and BLUE) as well as a synchronization signal. The color mode produces a display of 256×240 picture elements (PIXELS), each of which is three bits of information. The XTRA shift register data might be used for other functions like highlighting or blinking control but it was not utilized in

this application. The display convention is that the least significant bit of a display byte is the first bit to be displayed as the electron beam moves from left to right. Consecutive higher order bits follow in sequence. The RED bytes are the even addressed bytes of the "A" RAM. The GREEN bytes are the odd addressed bytes of the "A" RAM. The BLUE bytes are the even addressed bytes of the "B" RAM. The XTRA bytes are the odd addressed bytes of the "B" RAM. For each pixel of a color mode display, three bits of video information are required, one from each of the RED, GREEN, and BLUE shift registers. Thus to program the display it is useful to have a map of which memory bits map into which screen pixels. Figure 6 shows such a mapping for the color mode display.

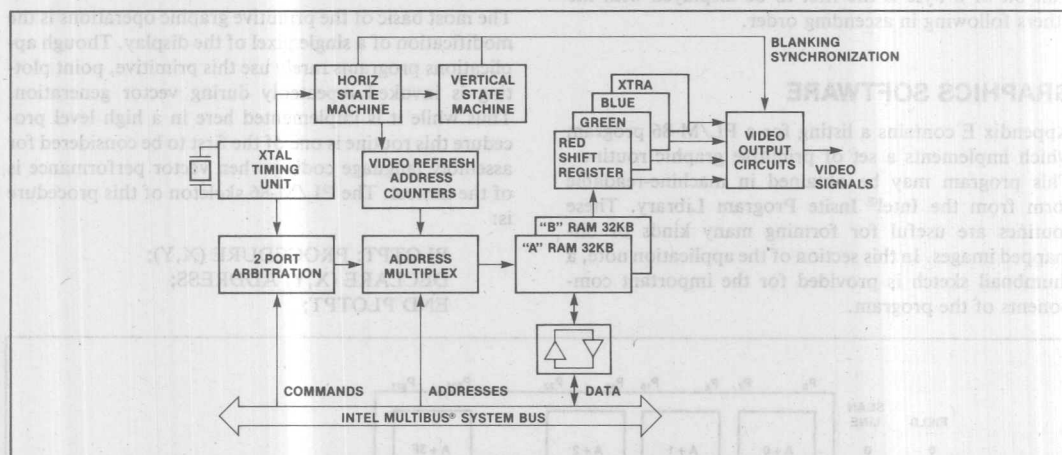


Figure 5. Block Diagram of Frame Buffer

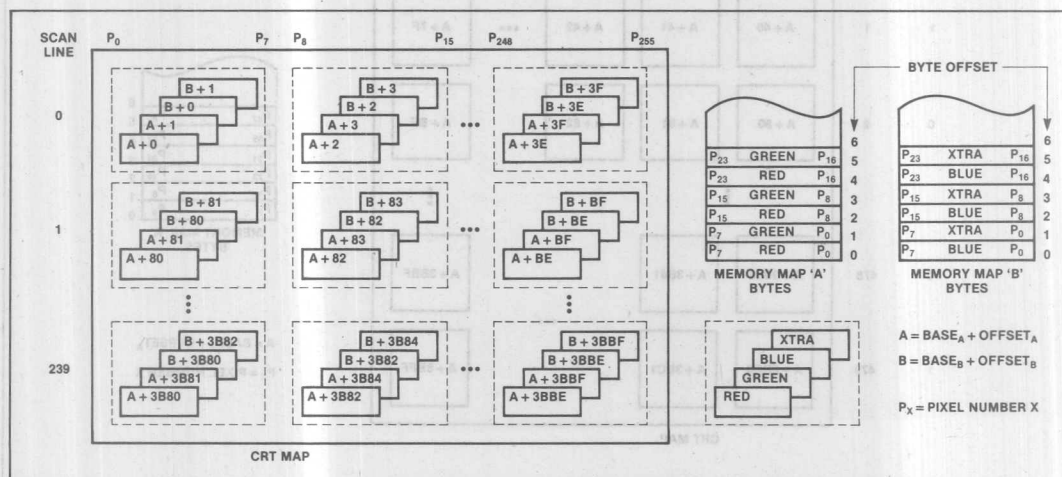


Figure 6. Color Byte Locations

Black/White (Monochromatic) Mode

To select monochromatic operation, the COLOR jumper must be installed in the Black/White position. When this is done the display changes to a 512×480 pixel format where each pixel is represented by a single bit from the frame buffer RAM. All the bytes to represent a full image are contained in the "A" RAM. With the duplication of some output circuits, the "B" RAM could produce another image and the board would be capable of supplying data to two independent monitors simultaneously.

Figure 7 shows the mapping of bytes of the frame buffer into pixel groups on the display. Again the least significant bit of a byte is the first to be displayed with the others following in ascending order.

GRAPHICS SOFTWARE

Appendix E contains a listing for a PL/M-86 program which implements a set of primitive graphic routines. This program may be obtained in machine-readable form from the Intel® Insite Program Library. These routines are useful for forming many kinds of bit-mapped images. In this section of the application note, a thumbnail sketch is provided for the important components of the program.

Throughout the discussion, the X,Y coordinates (to specify points, endpoints of lines, and vertices of polygons) are implicitly in the appropriate range for the type of display, color or monochrome, which has been selected with the COLORMODE switch. When COLORMODE is set to the Boolean TRUE, the range of X is 0 to 255 and the range of Y is 0 to 239. When COLORMODE is set to FALSE then the range of X is 0 to 511 and the range of Y is 0 to 479. As an added protection, particularly for the interactive use, run-time checking is performed on all X,Y coordinates passed as actual parameters to the graphic primitive routines (see PERROR procedure in the listing, Appendix E, line 213).

Point Plotting

The most basic of the primitive graphic operations is the modification of a single pixel of the display. Though applications programs rarely use this primitive, point plotting is invoked repeatedly during vector generation. Thus while it is implemented here in a high level procedure this routine is one of the first to be considered for assembly language coding when vector performance is of the utmost. The PL/M-86 skeleton of this procedure is:

```
PLOTPT: PROCEDURE (X,Y);
DECLARE (X,Y) ADDRESS;
END PLOTPT;
```

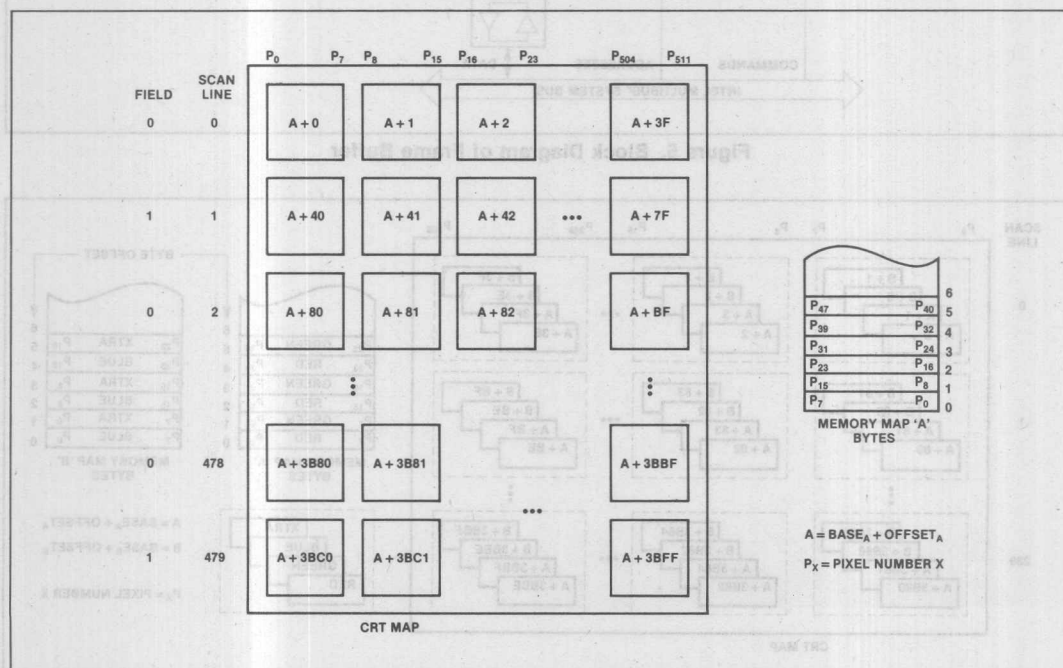


Figure 7. Monochrome Byte Locations

The procedure is contained in Appendix E starting at line number 226. The X and Y parameters specify the pixel address of the frame buffer that is to be modified. Appropriate shift and mask operations convert X and Y into a word address, PADR, which indexes into the frame buffer. The least significant four bits of the X-coordinate specify the bit position within the word to be modified.

A global byte variable PLOTMODE (Appendix E, line number 36) controls the operation to be performed on the specified pixel. The PLOTMODE operations are coded:

PLOTMODE Operation

```
0:      pixel = pixel OR 1;
        /* Set pixel */
1:      pixel = pixel AND 0;
        /* Clear pixel */
2:      pixel = pixel XOR 1;
        /* Complement pixel */
3...255: /* undefined */
```

Line Drawing

Chapter 2 of Newman and Sproull¹ is devoted to "Point Plotting Techniques" and the algorithm used to generate lines in this system is described on page 26. Bresenham's algorithm for implementing the digital differential analyzer (DDA) is coded in procedure DDA (Appendix E, line number 242). The PL/M-86 skeleton is:

```
DDA: PROCEDURE (X1,Y1,X2,Y2);
DECLARE (X1,Y1,X2,Y2) ADDRESS;
END DDA;
```

X1,Y1 and X2,Y2 are the endpoints of the line to be drawn by DDA. First the coordinates are sorted so that $Y2 \geq Y1$; Then the absolute value of the differences in X and Y coordinates (see Figure 8, DELX and DELY) are recorded in DELX and DELY (and twice those values in DELX2 and DELY2).

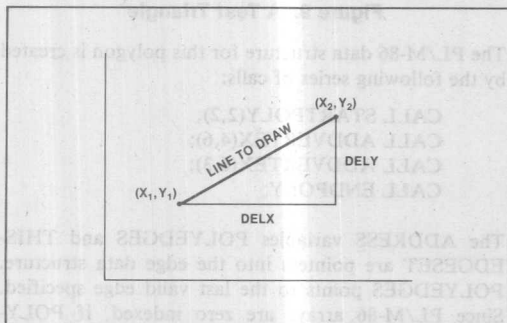


Figure 8. DELX and DELY

Three special cases are handled. A line whose endpoints are identical ($X1 = X2$ and $Y1 = Y2$) plots as a single pixel at $X1, Y1$. Horizontal lines ($DELY = 0$) and vertical lines ($DELX = 0$) are handled by simple loops which draw those special cases more efficiently.

If the line is not one of the three special cases described above, it is checked to determine whether the DELX or DELY component of the slope of the line is greater and an appropriate loop is initiated. The variable REM carries the remainder (or "e" error as in Newman and Sproull) for the incremental process which computes where each scan line of the display intersects the line segment being drawn. REM is a measure of the error generated as the points of a line are quantized for the frame buffer.

Scan Line Inking

Much of the power of raster graphic displays comes from their ability to display area-filled rectangular and polygonal areas. Though calligraphic and storage display devices can surpass raster displays in vector generation speed they lack the ability to fill areas or lend tone (and optionally color) to drawn areas.

The basic operation used to fill solid rectangular and polygonal areas is the inking (with a selected pattern, and possibly color) of the pixels on each visible scan line which comprises the image. The procedure INKLINE (Appendix E, line number 319) performs this operation. Its PL/M-86 skeleton is:

```
INKLINE: PROCEDURE (X1,X2,YSCAN);
DECLARE (X1,X2,YSCAN) ADDRESS;
END INKLINE;
```

X1 and X2 are the X coordinate extremes for the line of pixels to be drawn on the YSCAN scan line of the display. As might be expected, YSCAN has a range of 0 to 239 in color mode and 0 to 479 in black/white mode. First, X1 and X2 are sorted such that $X1 \leq X2$. Then a LEFTMASK and RIGHTMASK are formed which are used to perform partial word modifications which may occur at the left and right edge words of the scan line. Here the repeated shift of the 8086 is utilized, a very useful operation for many graphic primitive functions. XWORD is a count of the number of whole words of pixels which need to be modified. If XWORD=0, the pixels at X1 and X2 are included in the same word of frame buffer memory.

PLOTMODE affects the modification in much the same way as it did in PLOTPT, except that in INKLINE the

modification is word oriented rather than pixel oriented. Thus,

```
PLOTMODE Operation
0:      word = word OR 0ffffh;
        /* Set a word of pixels */
1:      word = word AND 00000h;
        /* Clear a word of pixels */
2:      word = word XOR 0ffffh;
        /* Complement a word of pixels */
3...255: /* undefined */
```

Rectangle Plotting

Rectangular areas of the image are modified by the PL/M-86 procedure PLOTRECT (Appendix E, line number 365) whose skeleton is:

```
PLOTRECT: PROCEDURE (X0,Y0,X1,Y1);
DECLARE (X0,Y0,X1,Y1) ADDRESS;
END PLOTRECT;
```

X0,Y0 and X1,Y1 are a pair of vertices which describe the endpoints of one of the diagonals of the rectangle to be filled. It is a straightforward process to fill the specified rectangle by calling INKLINE repeatedly, once for each visible scan line of the rectangle. PLOTRECT may be used to fill any rectangle whose edges are orthogonal to the coordinate axes. By clever control of the PLOTMODE, PLOTRECT can be used to fill, clear, and complement rectangular regions of the display.

Polygon Area Filling

Solid area filling of polygonal areas is a much more complicated task than that just explained for rectangles. Chapter 16 of Newman and Sproull¹ entitled "Solid Area Scan Conversion" outlines a number of alternative methods for area filling polygons.

A polygon may be represented as an ordered set of vertices; or as an ordered set of line segment edges. The PL/M-86 program in Appendix E records polygons in a data structure which is edge-oriented but provides a set of procedures to access that data structure which is vertex-oriented. This approach allows a user to traverse the vertices of a polygon in a natural fashion when filling the data structure. The fundamental polygon data structure (Appendix E, line numbers 23 through 35) is represented by the following PL/M-86 statements:

```
DECLARE EDGELIMIT LITERALLY '32';
DECLARE (X0(EDGELIMIT),Y0(EDGELIMIT))
ADDRESS;
DECLARE (X1(EDGELIMIT),Y1(EDGELIMIT))
ADDRESS;
DECLARE (POLYEDGES,THISEDGESET)
ADDRESS;
```

The vertex-oriented procedures are fairly self-explanatory from their PL/M-86 skeletons:

```
STARTPOLY: PROCEDURE (X,Y);
/* Appendix E, line number 409 */
DECLARE (X,Y) ADDRESS;
END STARTPOLY;
```

```
ADDVERTEX: PROCEDURE (X,Y);
/* Appendix E, line number 418 */
DECLARE (X,Y) ADDRESS;
END ADDVERTEX;
```

```
ENDPOLY: PROCEDURE;
/* Appendix E, line number 439 */
END ENDPOLY;
```

```
NEWEDGESET: PROCEDURE (X,Y);
/* Appendix E, line number 428 */
DECLARE (X,Y) ADDRESS;
END NEWEDGESET;
```

To demonstrate how these procedures are used and how the data structure is filled, consider Figure 9, A Test Triangle. The triangle has vertices (2,2), (4,6), and (6,3). Alternatively the triangle could be represented by a polygon with edges whose endpoints are ((2,2),(4,6)), ((4,6),(6,3)), and ((6,3),(2,2)).

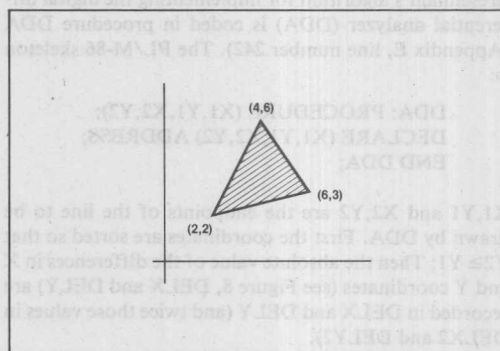


Figure 9. A Test Triangle

The PL/M-86 data structure for this polygon is created by the following series of calls:

```
CALL STARTPOLY(2,2);
CALL ADDVERTEX(4,6);
CALL ADDVERTEX(6,3);
CALL ENDPOLY;
```

The ADDRESS variables POLYEDGES and THISEDGESET are pointers into the edge data structure. POLYEDGES points to the last valid edge specified. Since PL/M-86 arrays are zero indexed, if POLYEDGES = 2 then three edges have been specified.

To accommodate a more complicated set of polygonal images one can consider a solid area image to be described by several distinct sets of polygon edge data. These edge sets might, for instance, describe the outer and inner edges of an image such as the "O" shape of Figure 10. As the data structure is filled, THISEDGESET points to the first edge of this (the current) edge set of the polygon. For the graphical object in Figure 11, Unusual Shape, the edge sets of this polygon could be specified by the following series of PL/M-86 calls.

The outermost edge set.

```
CALL STARTPOLY(2,2);
CALL ADDVERTEX(2,12);
CALL ADDVERTEX(12,12);
CALL ADDVERTEX(12,2);
```

The triangular inner edge set.

```
CALL NEWEDGESET(3,6);
CALL ADDVERTEX(5,3);
CALL ADDVERTEX(11,6);
```

The rectangular inner edge set.

```
CALL NEWEDGESET(10,7);
CALL ADDVERTEX(10,10);
CALL ADDVERTEX(4,10);
CALL ADDVERTEX(4,7);
CALL ENDPOLY;
```

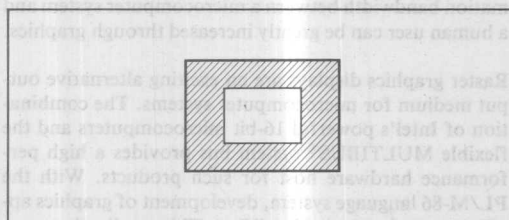


Figure 10. "O" Shape

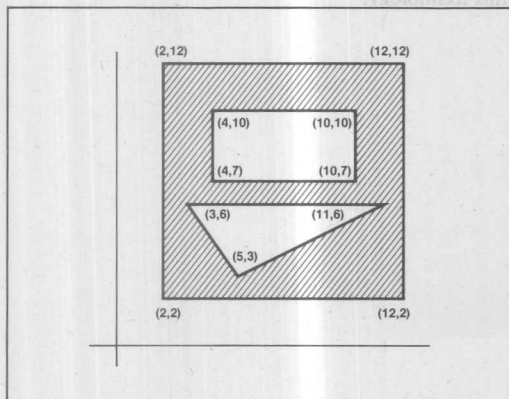


Figure 11. Unusual Shape

It is not important in which direction the vertices are visited when describing an edge set. Clockwise or counterclockwise order is acceptable. Whichever order has been selected must then be maintained for all the vertices on any particular edge set.

Filling a polygon is then the process of finding, for each visible scan line of the polygon, the intersection of all edges of the polygon with the scan line. These intercepts are sorted and accumulated in pairs of ascending X values. The first X-intercept on a scan line causes scan line inking to begin and the second X-intercept causes scan line inking to stop. This process may be repeated for several line segments to form all the visible detail on a single scan line. The process is repeated for all the visible scan lines of the polygon. Only memory constraints limit the size of the polygon edge structure, and hence the complexity (number of edges) of polygons which can be plotted.

Figure 12 is a state diagram for the proper sequences of calls to the vertex-oriented procedures for filling the polygon data structure. A call to STARTPOLY provides the first vertex of the first edge set of the polygon being described. Any edge set must contain at least three edges since a triangle is the simplest polygon which can be represented and displayed. Thus a minimum of two additional calls to ADDVERTEX must follow. Notice that ADDVERTEX can be repeatedly called to construct arbitrarily complex edge sets. Either ENDPOLY (if this is the last edge set) or NEWEDGESET is invoked to terminate an edge set. ENDPOLY also terminates the polygon data structure. NEWEDGESET terminates the edgset just constructed and specifies the first coordinates of a new edge set which is to be started. The sum of the number of calls to STARTPOLY, ADDVERTEX, and NEWEDGESET must be less than or equal to the EDGELIMIT value.

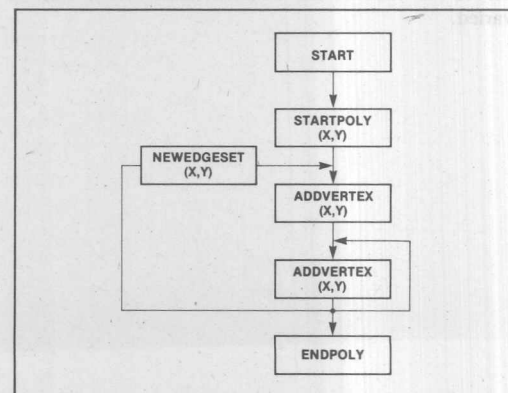


Figure 12. State Diagram for Vertex-Oriented Procedure Calls

Indeed, the PL/M-86 program in the Appendix contains far more than just the primitive graphics operations described above. A simple user interface is provided to allow interactive access to a few illustrative operations.

DEMONSTRATION SOFTWARE

In Appendix E is a demonstration program which illustrates how an application can be constructed. This program has a set of predefined graphic operations which can be invoked to produce some graphic displays. It also allows the user to interact with the system and dynamically construct polygons with a simple input device called a turtle.

At either the 'turtle' or the 'main' level of the program typing 'h' will generate a help file of the program options.

Predefined Operations

The predefined operations of the program are functions which do useful things such as clear the frame buffer, set graphic modes, establish the polygon data, plot color test patterns, etc. The main level help file lists all the functions which may be invoked.

The Turtle

The demonstration program contains the simplest of all graphic input devices, the turtle. By typing a 't' to the main command level, the turtle level is entered. In the turtle section of the program, a small rectangle is displayed which signifies the present location of the turtle. The turtle is commanded by the user through single character directives, 'u' for up, 'd' for down, 'l' for left, and 'r' for right; an effective though somewhat tedious, process. The crawlspeed, the distance which the turtle moves for each of the direction commands, may be varied.

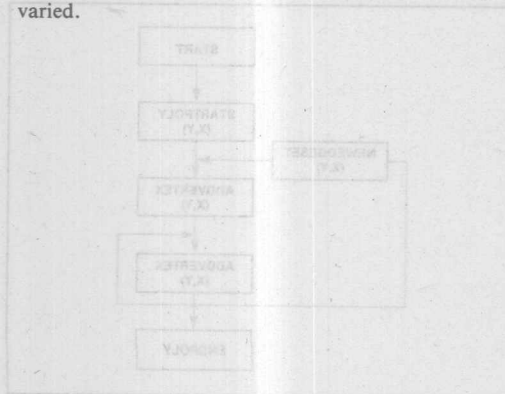


Figure 12. State Diagram for Vertex-Oriented Procedure Calls

Additional single character commands allow the polygon procedures to be invoked, each one adding a vertex to the polygon data structure based on the current location of the turtle on the graphics display. As each point is added the edges being added to each edgeset of the polygon are displayed on the graphics screen. When the exit key, 'e' is typed the polygon data structure is completed, the turtle erased, and the main command loop entered. The specified polygon may then be plotted.

CONCLUSION

Intel's iSBC 86/12A and iSBC 86/05 (8 MHz performance) single board computers, based on the 8086 microprocessor, bring minicomputer performance to the MULTIBUS® system bus. Supporting an address space of 1 million bytes, these microcomputers allow memory-intensive applications which were previously limited by less capable processors. 16-bit data transfers also afford dramatic performance improvements, for applications like computer graphics, over that of earlier microcomputers.

Many products can benefit from raster graphic display devices. Electronic graphic displays allow information to be presented in a more natural manner than can be done with simple character-based displays. The information bandwidth between a microcomputer system and a human user can be greatly increased through graphics.

Raster graphics displays are an exciting alternative output medium for microcomputer systems. The combination of Intel's powerful 16-bit microcomputers and the flexible MULTIBUS® system bus provides a high performance hardware host for such products. With the PL/M-86 language system, development of graphics application software is simplified. This application note demonstrates how easy and economical it is to utilize this technology.

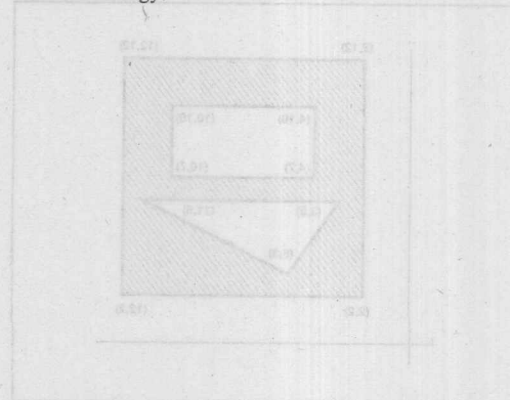


Figure 13. Unusual Shape

APPENDIX A

MULTIBUS® FRAME BUFFER HARDWARE DESCRIPTION

The hardware for the MULTIBUS® Frame Buffer consists of several sections of logic:

- Two Port Memory Control Logic
- Address Generation for Video Refresh
- Synchronization and Blanking Logic
- MULTIBUS® Interface Logic
- Video Output Logic

These will be discussed individually. Please refer to the schematics in Appendix B.

Two Port Memory Control Logic

The two port memory control logic manages the 64K byte dynamic RAM array. It provides video refresh data to the display, read/write access by the MULTIBUS® master, and synchronous management of refresh cycles for the dynamic RAM chips.

An Intel® 8224 Clock Generator and Driver with a 10.0 MHz crystal provides the timing reference clock (PIXCLK) for the display controller. A 74S163 counter divides down PIXCLK to form 16 states called SF...S0. These states are partially decoded by a 3205 one-of-eight decoder and after additional gating are resynchronized with a 74S374 pipeline register to form the seven basic control signals on the board. The function of each of these signals is described as follows:

STRAS/	Start generating RAS/ for the memory array.
STROW/	Start generating the row address enable signal (ROWEN).
STCAS/	Start generating CAS/ for the memory array.
NDCTL/	End control signals.
LATCH	Latch memory data for return to the MULTIBUS® system bus.
SRTIME	Load video shift register data when enabled by SRENb.
ADV	Advance address counters and state machines.

The two port memory control logic timing diagram included in Appendix C describes the generation of these

signals. States S0 through S7 are reserved for the use of the video refresh port and for dynamic RAM chip refresh. States S8 through SF are reserved for possible MULTIBUS® access. Thus it is always possible for the processor to access the memory between video or dynamic RAM refresh cycles.

Address Generation for Video Refresh

The video refresh address counter consists of four 74LS163 binary counters. The 14 bit address which is formed selects two 16 bit words in the frame buffer RAM, one from the "A" RAM and one from the "B" RAM. A multiplexing scheme on the address bus AD...A0 presents the MULTIBUS® addresses ADRE/...ADRI/ when BUS is active or the video refresh addresses, Y8...Y0 and X4...X0 when BUS is inactive. Obviously Y8...Y0 corresponds to the scan line address and X4...X0 corresponds to the 16 bit word within the scan line.

A 74S74 flip flop generates the signal ODD (ODD=0 for field 0, ODD=1 for field 1). The X-Y address counter latches the value of ODD at the beginning of each field of the display and Y0 carries the latched value of ODD for the duration of the field.

Synchronization and Blanking Logic

Two finite state machines generate all synchronization and blanking signals. In addition these machines control the advance of the video refresh address counters and sequence the video shift register data. The signal COLOR is an input to each of these machines which changes their operation to accomplish the appropriate display format.

One of these machines is the horizontal state machine and it consists of two 74LS163 counters and an Intel® 2716 PROM. One of the counters serves as the state counter and the other as a duration counter for the amount of time to remain in the current state. The horizontal control PROM examines the current horizontal machine state (HCP3...HPC0), the condition of the vertical state machine (VBLK/), and COLOR to determine which control signals to generate in the next machine state and how long the signals are to remain.

The horizontal state machine PROM generates the following signals:

PH12	Duration control bit
PH2	Duration control bit
PH1	Duration control bit
PNEXTV	Next vertical count, occurs at twice the horizontal line rate
PREFR/	Refresh enable for 3242 and two port logic
PSRENB	Shift register enable to load data, address counter enable
PHBLK/	Horizontal blanking control
PHSYNC/	Horizontal synchronization (drive) control

The prefix "P" on each of the above signals indicates that the PROM is a source of the signal. Since all outputs of the PROMs are allowed a full ADV cycle (16 PIXCLK periods) before they are required to be stable there is no critical access time requirement for the 2716s. With a 10.0 MHz clock, 1.6 microseconds are available.

A 74S374 pipeline register, clocked by the rising edge of ADV, holds stable control signals throughout the next ADV period.

The vertical state machine is constructed in the same manner as the horizontal one except that an additional 74LS163 counter is used to extend the duration capability of the machine. The vertical control PROM produces the following signals:

PVBLK/	Vertical blanking control
PVSYNC/	Vertical synchronization (drive) control
PV224	Duration control bit
PV16	Duration control bit
PV8	Duration control bit
PV4	Duration control bit
PV2	Duration control bit
PV1	Duration control bit

As the vertical state counter reaches the terminal count (VPC3...VPC0=15) the signal FIELD0/ is asserted which causes the horizontal state to be set to zero. This guarantees that the phase of the two machines is in synchronism. FIELD0/ also preconditions the signal ODD to zero in preparation to entering the first of the two fields of each frame.

The contents of the two PROMs required to interface to an NEC Model JC-1202 DH color monitor or a Ball Brothers TTL-120 monochrome display are included in Appendix D. The timing diagrams in Appendix C represent the PROM information in a more graphical man-

ner. Using these two control PROMs, one of the two displays is generated:

262 line color display, non-interlaced, with 240 visible scan lines, 256 pixels per scan line, 4 bits/pixel.

525 line monochrome display, interlaced 2:1, with 480 visible scan lines, 512 pixels per scan line, 1 bit/pixel.

Of course with other PROM codings and crystal frequencies, different display formats and monitor control signals are possible.

MULTIBUS® Interface Logic

The MULTIBUS® architecture's full megabyte address capability allows direct mapping of the 64K bytes of the frame buffer RAM on to the system bus. In similar display systems designed for traditional minicomputer bus structures or early microcomputers some form of address mapping or bank selection was usually necessary since the address space of those machines was limited (64K to 256K bytes).

Two 3205 decoders examine the MULTIBUS® address lines ADR13/...ADRF/ and the RAM inhibit signal INH1/. A 74S32 gate and two jumpers allow each 32K byte frame buffer RAM ("A" and "B") section to be placed at any 32K byte boundary in the one megabyte system address space.

Three 8286 bus transceivers are used in the byte swapping configuration to buffer the onboard 16 bit wide bidirectional data bus from the MULTIBUS® lines DATF/...DAT0/.

Four 74S74 flip flops are sequenced by the two port control logic to synchronize MULTIBUS® requests to the on-board timing. The four stage pipeline is initiated with the receipt of memory read (MRDC/) or memory write (MWTC/) commands. If the board is selected (BDSEL/ low) when CMD strobes the first flip flop, service to the request from the MULTIBUS® system bus is begun. On the rising edge of SRTIME (start of state S7) an arbitration flip flop is strobed to synchronize the bus request to the on-board clock. One state later a second arbitration flip flop is clocked to minimize the possibility of error in the synchronization. This third flip flop produces the signal BUS which is aligned with state S8 and specifies to the two port control logic that a MULTIBUS® access is being serviced. The memory array is sequenced just as for a video refresh cycle. In the case that the CMD is actually a write operation, one or more of the signals WRRED/, WRGRN/, WRBLU/, and WRXTRA/ are

activated to the memory array and cause the cycle to be either a single byte write to the high or low byte of the array or a full 16 bit store operation.

At the end of any MULTIBUS® port cycle the signal LATCH is produced which performs two functions. First, LATCH opens a 74S373 transparent latch and then closes it to capture the data on the RAM data out lines. In the case of a MULTIBUS® read, this data is returned to the bus, and in the case of a write, it is discarded. Additionally, the falling edge of LATCH clocks a flip flop which clears the bus synchronization flip flops and forms XACK/, thus informing the MULTIBUS® master that the transfer has been acknowledged.

Video Output Logic

The video output section of the display controller forms the three primary signals required by conventional raster scan display monitors: blanked video data (RED, GREEN, and BLUE for color), horizontal drive and vertical drive.

When used to control a monochrome display, the video data from the frame buffer memory may be displayed in normal or reversed mode. In normal mode a logical one

in the frame buffer generates white pixels and a logical zero generates black pixels. In reversed video mode a logical one in the frame buffer memory generates black pixels and vice versa.

Regardless of the video mode chosen, the pixel stream is properly blanked (by HBLK/ and VBLK/) before it is presented to the monitor. HBLK/ blanks the video information during the horizontal retrace time and VBLK/ blanks the video during the vertical retrace interval.

A final 74S175 register, duplicated for the color and monochrome output sections, removes any skew and jitter between the video pixels and the synchronization signals and also affords the user the option of selecting the polarity of the synchronization signals. In the case of the NEC color monitor, a combination of horizontal and vertical synchronization signals is required. Finally, the selected output signals are each buffered by a 74128 line driver suitable for driving the low impedance (50 to 100 ohms) terminated lines which are preferred for transporting these signals. The 74128 drivers are actually 50 ohm drivers and for driving lines with higher characteristic impedances it is proper to add the appropriate series termination resistor.

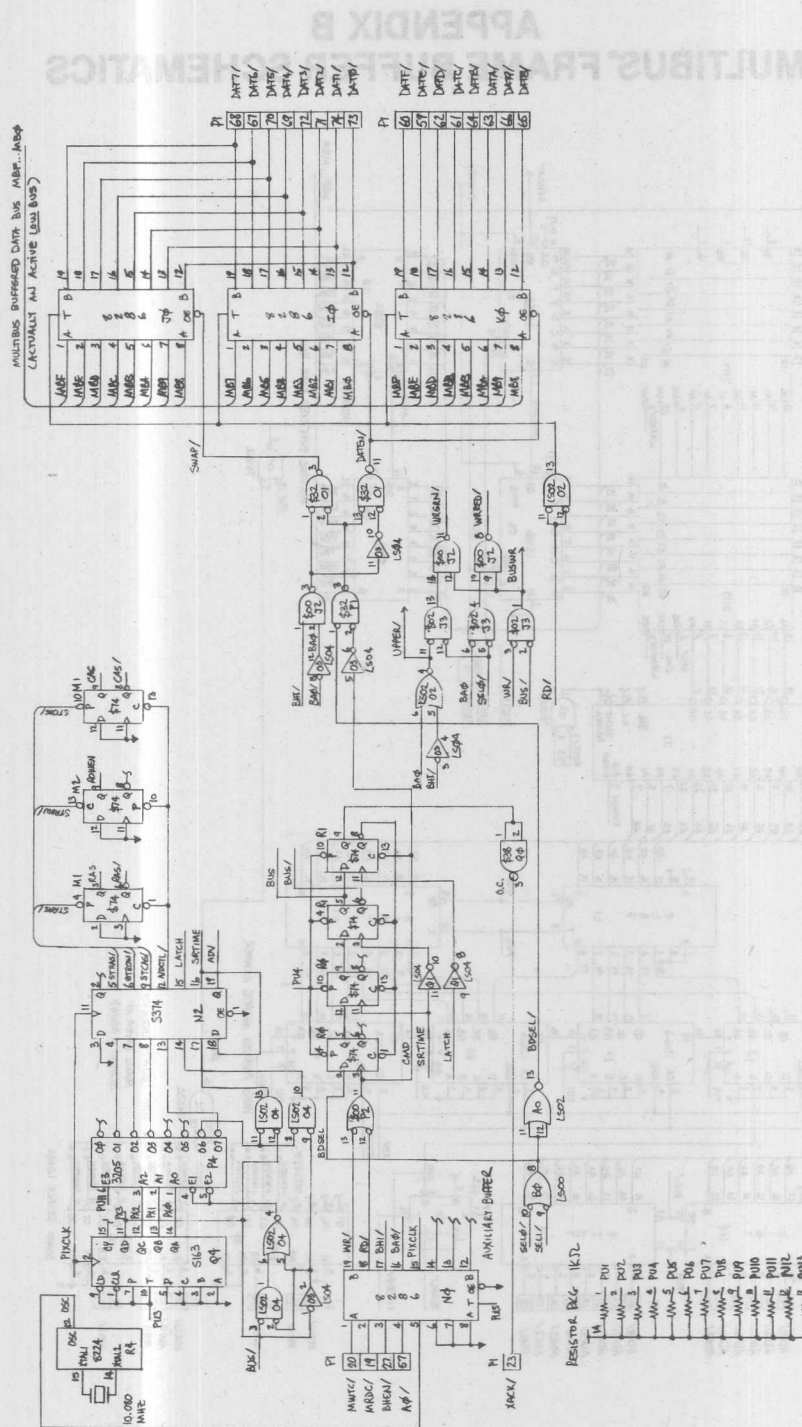
The vertical state machine is constructed in the same manner as the horizontal one except that an additional 74S163 counter is used to extend the duration capability of the machine. The vertical control PROM produces the following signals:

PVBLK/	Vertical blanking control
PVSYNC/	Vertical synchronization (drive) control
PV22S	Duration control bit
PV16	Duration control bit
PV8	Duration control bit
PV4	Duration control bit
PV2	Duration control bit
PV1	Duration control bit

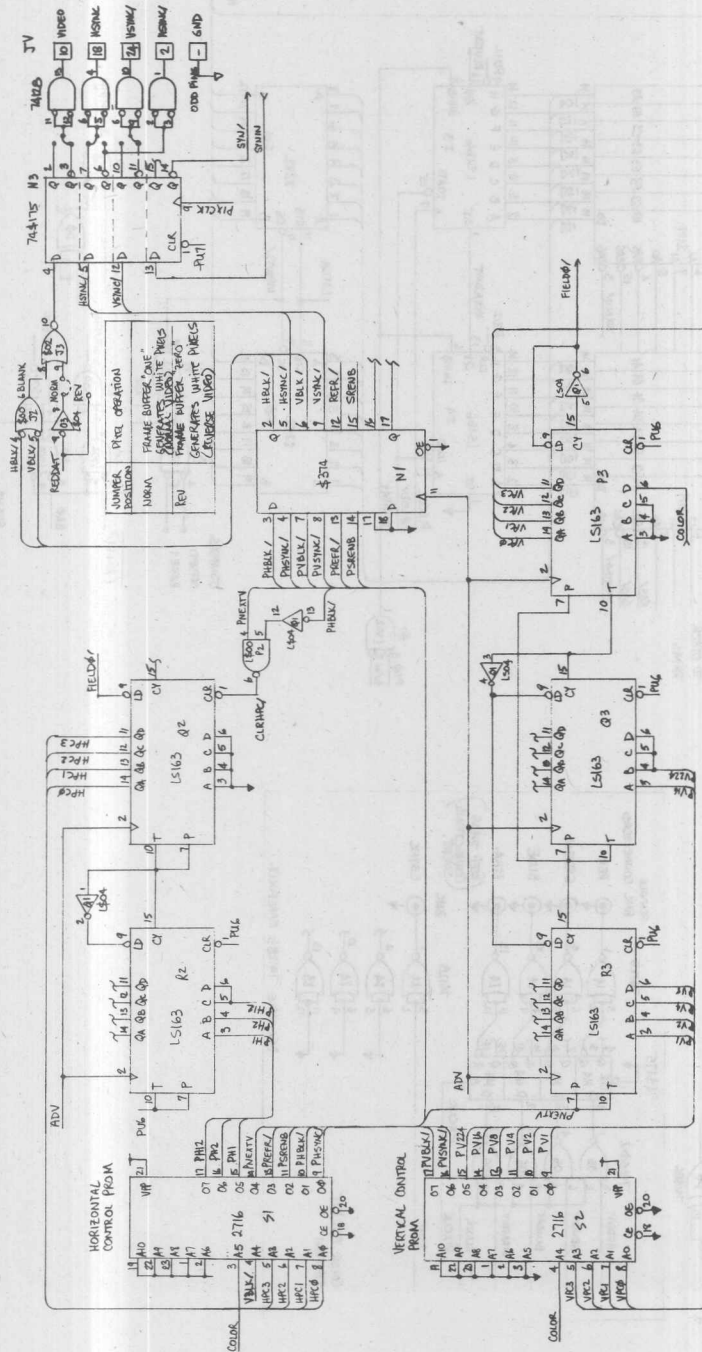
As the vertical state counter reaches the terminal count (VPC...VPC0 = 1) the signal FIELD0/ is asserted which causes the horizontal state to be set to zero. This guarantees that the phase of the two machines is in synchronism. FIELD0/ also preconditions the signal ODD to zero in preparation to entering the first of the two fields of each frame.

The outputs of the two PROMs required to interface to an NEC Model IC-1203 DH color monitor or a Ball-Brooks TTL-130 monochrome display are included in Appendix D. The timing diagrams in Appendix C represent the PROM information in a more graphical manner.

Frame Buffer Schematics (sheet 1 of 4)

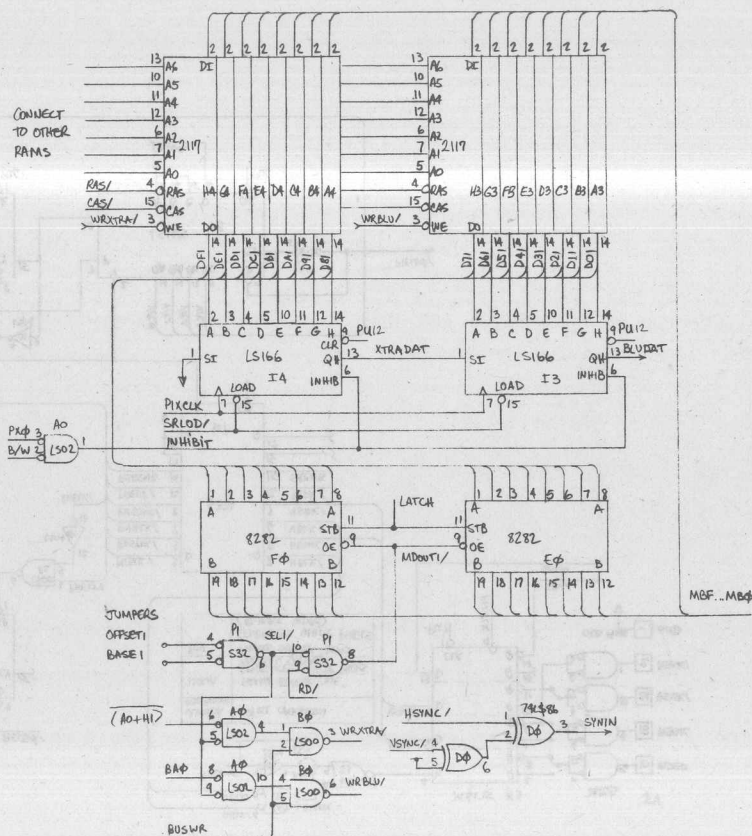


Frame Buffer Schematics (sheet 2 of 4)



Frame Buffer Schematics (sheet 3 of 4)

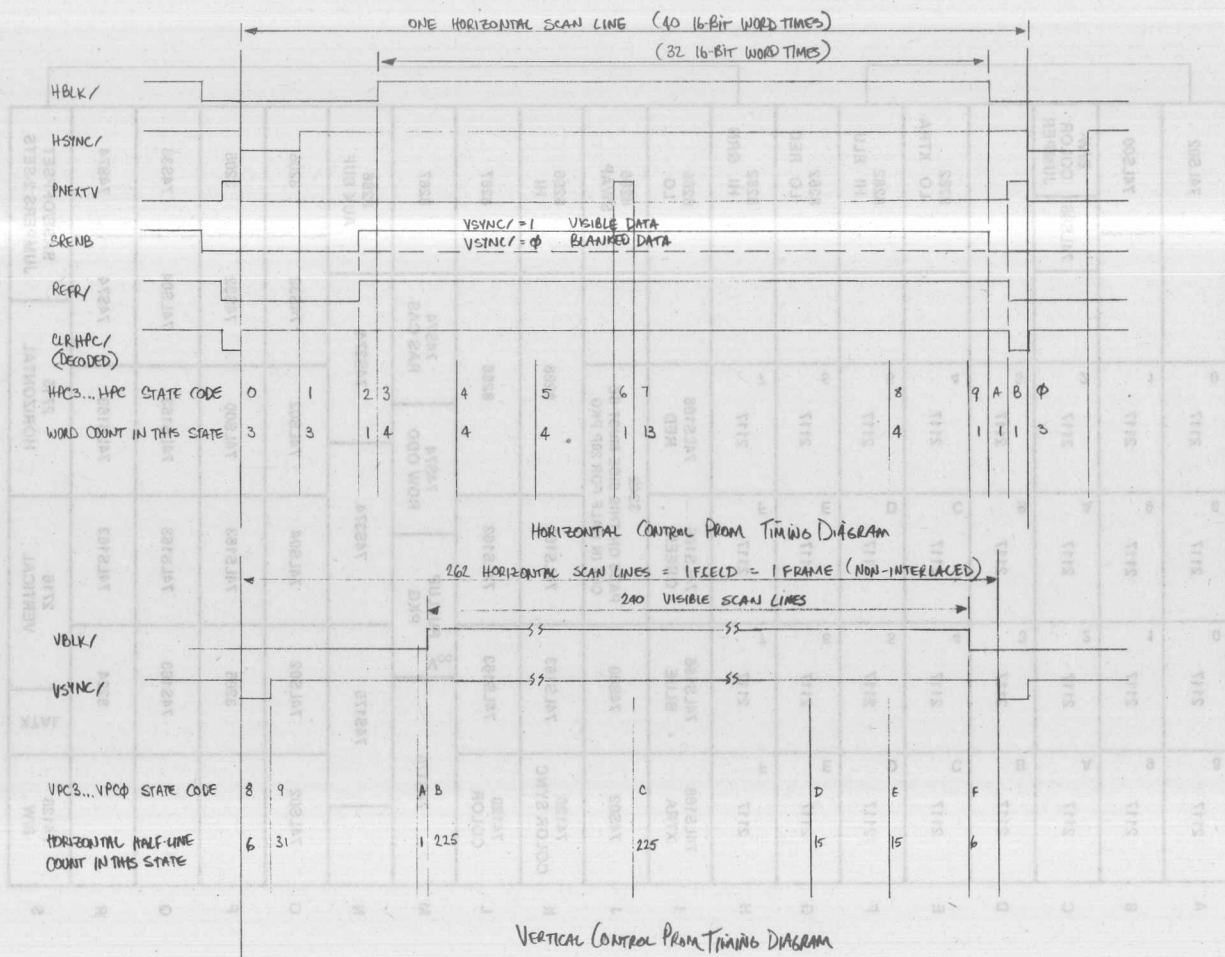
Frame Buffer Schematics (sheet 4 of 4)



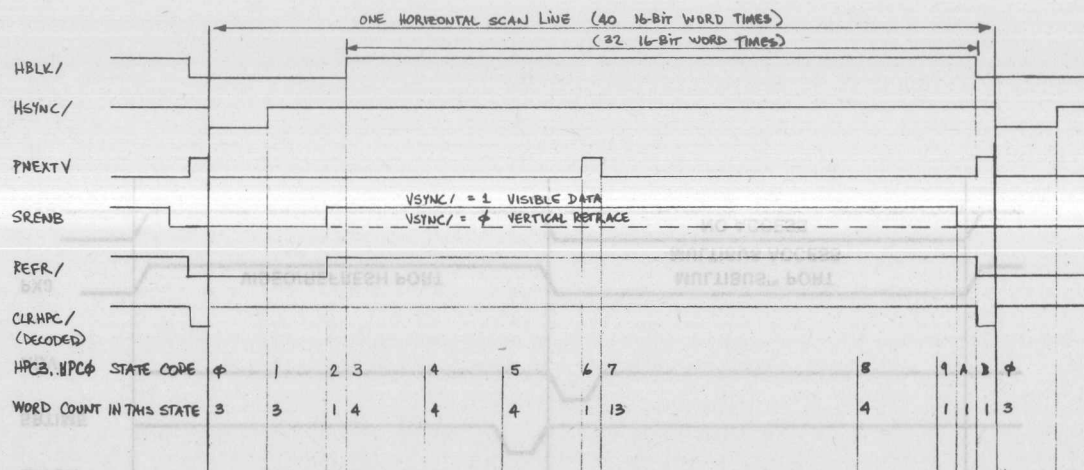
	5	4	3	2	1	0
A	2117 8	2117 0	2117 8	2117 0		74LS02
B	2117 9	2117 1	2117 9	2117 1		74LS00
C	2117 A	2117 2	2117 A	2117 2	74LS86	B/W-COLOR JUMPER
D	2117 B	2117 3	2117 B	2117 3		
E	2117 C	2117 4	2117 C	2117 4		8282 LO XTRA
F	2117 D	2117 5	2117 D	2117 5		8282 HI BLU
G	2117 E	2117 6	2117 E	2117 6		8282 LO RED
H	2117 F	2117 7	2117 F	2117 7		8282 HI GRN
I	74LS166 XTRA	74LS166 BLUE	74LS166 GREEN	74LS166 RED		8286 LO
J	74S02	74S00	3242 PADS ON THIS SIDE MUST BE CUT IN HALF FOR 28P PKG.			8286 SWAP
K	74128 COLOR SYNC	74LS163	74LS163	8286		8286 HI
L	74128 COLOR	74LS163	74LS163	8286		8287
M	74S175	V _{CC} PULLUP PKG	74S74 ROW ODD	74S74 RAS CAS		8287
N		74S175	74S374	74S374		8286 AUX BUF
O	74LS02	74LS02	74LS04	74LS02	74S32	3205
P		3205	74LS163	74LS00	74S32	3205
Q		74S163	74LS163	74LS163	74LS04	74S38
R		8224	74LS163	74LS163	74S74	74S74
S	74128 B/W	XTAL	2716 VERTICAL	2716 HORIZONTAL	BASE/OFFSET JUMPERS 2-SETS	

Frame Buffer Parts Layout (component side view)

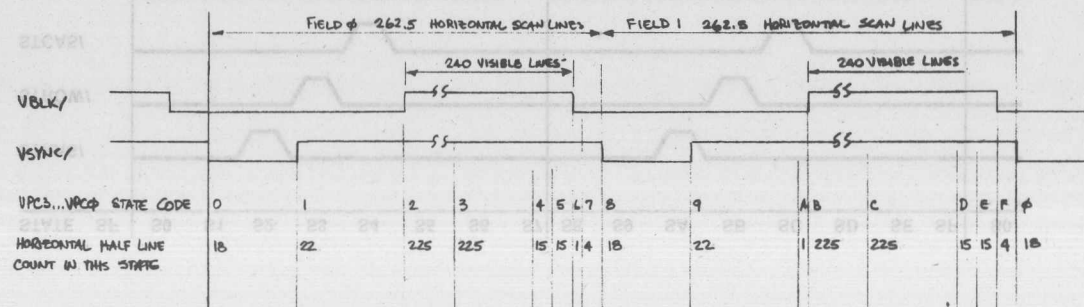
APPENDIX C TIMING DIAGRAMS



NEC Color Monitor Model JC-1202 DH(A) Timing Diagrams

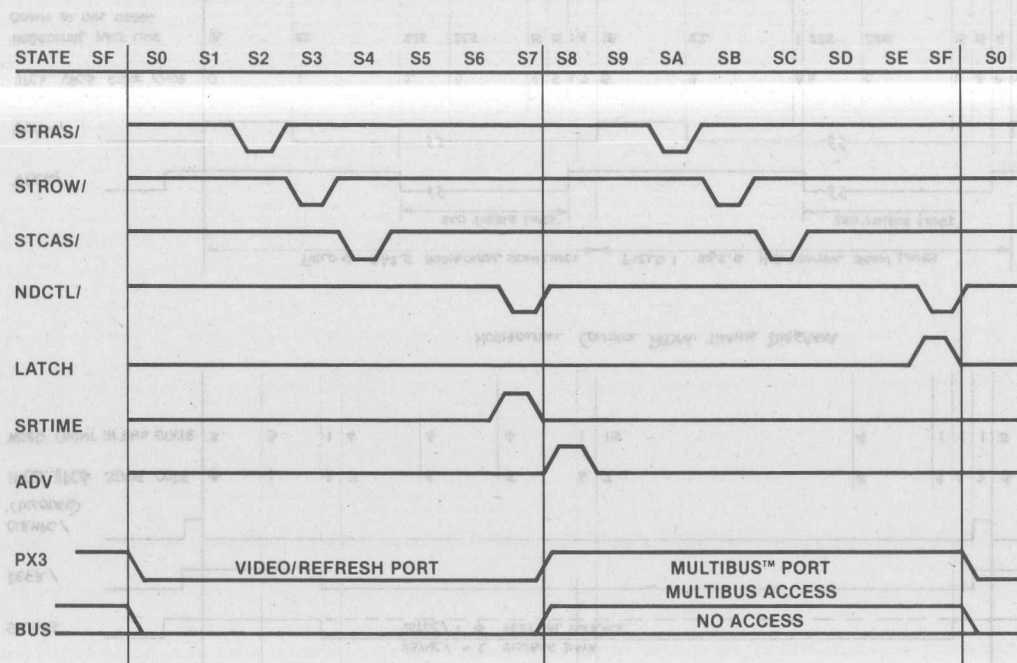


HORIZONTAL CONTROL FROM TIMING DIAGRAM



VERTICAL CONTROL FROM TIMING DIAGRAM

Black/White Ball Brothers TTL120 Timing Diagrams



Two Port Memory Control Logic Timing Diagram

APPENDIX D CONTROL PROM CODES

```

LOC  OBJ          LINE      SOURCE STATEMENT
      1:          Horizontal: PROM Codes for MULTIBUS Frame Buffer
      2
0000      3          org      0
      4:          +----- PH12
      5:          +----- PH2
      6:          +----- PH1
      7:          +----- PNEXTV
      8:          +----- PREFR/
      9:          +----- PSRENB
     10:          +----- PBLK/
     11:          +----- PHSYNC/
     12:          755A3210      PROM Data Output
     13:          +-----
     14:          +-----      ; required count for next state
     15:          -----      Black/White mode
0000 A0      16          db      10100000b      ; 3
0001 E9      17          db      11101001b      ; 1
0002 89      18          db      10001001b      ; 4
0003 88      19          db      10001011b      ; 4
0004 88      20          db      10001011b      ; 4
0005 E9      21          db      11101011b      ; 1
0006 78      22          db      01111011b      ; 13
0007 88      23          db      10001011b      ; 4
0008 E8      24          db      11101011b      ; 1
0009 E8      25          db      11101011b      ; 1
000A E8      26          db      11101011b      ; 1
000B B1      27          db      10110001b      ; 3
000C 00      28          db      00000000b      ; UNUSED
000D 00      29          db      00000000b      ; UNUSED
000E 00      30          db      00000000b      ; UNUSED
000F 00      31          db      00000000b      ; UNUSED
0010 A0      32          db      10100000b      ; 3
0011 E9      33          db      11101001b      ; 1
0012 80      34          db      10001101b      ; 4
0013 8F      35          db      10001111b      ; 4
0014 8F      36          db      10001111b      ; 4
0015 EF      37          db      11101111b      ; 1
0016 7F      38          db      01111111b      ; 13
0017 8F      39          db      10001111b      ; 4
0018 EF      40          db      11101111b      ; 1
0019 EF      41          db      11101111b      ; 1
001A E8      42          db      11101011b      ; 1
001B B1      43          db      10110001b      ; 3
001C 00      44          db      00000000b      ; UNUSED
001D 00      45          db      00000000b      ; UNUSED
001E 00      46          db      00000000b      ; UNUSED
001F 00      47          db      00000000b      ; UNUSED
     48          select
     49          -----      Color mode
0020 A0      50          db      10100000b      ; 3
0021 E9      51          db      11101001b      ; 1
0022 89      52          db      10001001b      ; 4
0023 88      53          db      10001011b      ; 4
0024 88      54          db      10001011b      ; 4
0025 E9      55          db      11101011b      ; 1
0026 78      56          db      01111011b      ; 13
0027 88      57          db      10001011b      ; 4
0028 E8      58          db      11101011b      ; 1
0029 E8      59          db      11101011b      ; 1
002A E9      60          db      11101001b      ; 1
002B B1      61          db      10110001b      ; 3
002C 00      62          db      00000000b      ; UNUSED
002D 00      63          db      00000000b      ; UNUSED
002E 00      64          db      00000000b      ; UNUSED
002F 00      65          db      00000000b      ; UNUSED
0030 A0      66          db      10100000b      ; 3
0031 E9      67          db      11101001b      ; 1
0032 80      68          db      10001101b      ; 4
0033 8F      69          db      10001111b      ; 4
0034 8F      70          db      10001111b      ; 4
0035 EF      71          db      11101111b      ; 1
0036 7F      72          db      01111111b      ; 13
0037 8F      73          db      10001111b      ; 4
0038 EF      74          db      11101111b      ; 1
0039 EF      75          db      11101111b      ; 1
003A E9      76          db      11101001b      ; 1
003B B1      77          db      10110001b      ; 3

```

```

LOC OBJ      LINE      SOURCE STATEMENT
000C 00      78      db      00000000b      : UNUSED
000D 00      79      db      00000000b      : UNUSED
000E 00      80      db      00000000b      : UNUSED
000F 00      81      db      00000000b      : UNUSED
          82
          83 :      PROM Input Variables
          84 :
          85 :      COLOR - A5
          86 :      VBLK/ - A4
          87 :      HPC1 - A3
          88 :      HPC2 - A2
          89 :      HPC1 - A1
          90 :      HPC0 - A0
0000      91      end      0

LOC OBJ      LINE      SOURCE STATEMENT
          1 :      Vertical PROM Codes for MULTIBUS Frame Buffer
          2 :      org      0
          3 :
          4 :      +----- PVBK/
          5 :      +----- PVSINC/
          6 :      +----- PV224
          7 :      +----- PV16
          8 :      +----- PV8
          9 :      +----- PV4
          10 :      +----- PV2
          11 :      +----- PV1
          12 :      76543210      PROM Data Outputs
          13 :
          14 :      +----- : required count for next state
          15 :      ----- Black/white mode

0000 2A      16      db      00101010b      : 22
0001 5F      17      db      01011111b      : 225
0002 DF      18      db      11011111b      : 15
0003 F1      19      db      11110001b      : 15
0004 F1      20      db      11110001b      : 15
0005 FF      21      db      11111111b      : 1
0006 7C      22      db      01111100b      : 4
0007 6E      23      db      01101110b      : 18
0008 2A      24      db      00101010b      : 22
0009 7F      25      db      01111111b      : 1
000A 5F      26      db      01011111b      : 225
000B DF      27      db      11011111b      : 225
000C F1      28      db      11110001b      : 15
000D F1      29      db      11110001b      : 15
000E FC      30      db      11111100b      : 4
000F 6E      31      db      01101110b      : 18
          32 : ----- Color mode
0010 00      33      db      00000000b      : unused
0011 00      34      db      00000000b      : unused
0012 00      35      db      00000000b      : unused
0013 00      36      db      00000000b      : unused
0014 00      37      db      00000000b      : unused
0015 00      38      db      00000000b      : unused
0016 00      39      db      00000000b      : unused
0017 00      40      db      00000000b      : unused
0018 21      41      db      00100001b      : 31
0019 7F      42      db      01111111b      : 1
001A 5F      43      db      01011111b      : 225
001B DF      44      db      11011111b      : 225
001C F1      45      db      11110001b      : 15
001D F1      46      db      11110001b      : 15
001E FA      47      db      11111010b      : 6
001F 7A      48      db      01111010b      : 6
          49 :
          50 :      Vertical PROM Input Variables
          51 :
          52 :      COLOR - A4
          53 :      VPC3 - A3
          54 :      VPC2 - A2
          55 :      VPC1 - A1
          56 :      VPC0 - A0
          57 :
          58 :      end      0

```


APPENDIX E

PL/M-86 GRAPHICS SOFTWARE LISTING

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE PIXEL

OBJECT MODULE PLACED IN pixel.OBJ

COMPILER INVOKED BY: tfl:plm86 pixel.src LARGE OPTIMIZE(0)

```

$ title('MULTIBUS Graphics Application Note')
$ date(26 Apr 81)

/*
This PL/M-86 program demonstrates the use of a number of
primitive graphics routines used to drive the MULTIBUS
raster graphics controller board.

The MULTIBUS frame buffer board is located at 00000H-0ffffH
and at 010000H-01ffffH in the megabyte address space.

There are five main areas to this program:
1. Declarations for variables and polygon data structure.
2. Low-level I/O procedures for the SBC-86/12A.
3. The graphics routines.
4. Turtle: a simple graphics input device.
5. The main body, mainly a simple command decoder.
*/
pixel:
do:

2 1 declare (x,y,size) address;
3 1 declare true literally '0ffh';
4 1 declare false literally '000h';
5 1 declare cr literally '00ah';
6 1 declare lf literally '00dh';

/* SBC-86/12A constants for 8251 USART Status and Data */
7 1 declare usstat literally '0dah';
8 1 declare usdata literally '0d8h';

/* 8251 status - receive buffer full,transmit buffer empty */
9 1 declare rxbf literally '002h';
10 1 declare txbe literally '001h';

/* limit to number of edges in a complex polygon */
11 1 declare edgelimt literally '32';

/* width/height limits (in pixels) for the CRT screen */
12 1 declare (screenwidth,screenheight) address;

/* an array for hexadecimal character conversions */
13 1 declare hexcode (*) byte initial ('0123456789ABCDEF');

/* text strings for console messages */

14 1 declare signon (*) byte initial
    (cr,lf,'MULTIBUS Graphics Application Note System ',
    '4/20/1981',cr,lf);

15 1 declare edgetitle (*) byte initial
    ('Edge: X0 Y0 X1 Y1 delx dely rem',cr,lf);

16 1 declare pointerror (*) byte initial
    ('Point out of bounds @ x,y: ');

17 1 declare turtlehelp (*) byte initial
/* help file for the turtle facility */
(cr,lf,
'a - AddVertex',cr,lf,
'c(s) - CrawlSpeed',cr,lf,
'd - TurtleDown',cr,lf,
'e - EndPolygon',cr,lf,
'h - Help',cr,lf,
'i - TurtleLeft',cr,lf,
'n - NewEdgeSet',cr,lf,
'r - TurtleRight',cr,lf,
's - StartPolygon',cr,lf,
'u - TurtleUp',cr,lf);

18 1 declare mainhelp (*) byte initial
/* help file for the main program loop */
(cr,lf,
'a(x)(y) - AddVertex @ x,y',cr,lf,
'b(c) - Select color c',cr,lf,
'c - Clear',cr,lf,
'e - EndPolygon',cr,lf,
'f(x)(y) - Fantasy @ x,y',cr,lf,
'h - Help',cr,lf,
'i(f) - Ink with font f',cr,lf,
'l - Outline Polygon',cr,lf,
'm(x) - Mode Selection x ',
'0 - or, 1- and not, 2- xor',cr,lf,
'n(x)(y) - NewEdgeSet @ x,y',cr,lf,
'p - FillPolygon',cr,lf,
'q - Toggle color/bw boolean',cr,lf,
's(x)(y) - Start Polygon @ x,y',cr,lf,
't - Turtle',cr,lf,
'1 - Setup Pattern 1',cr,lf,
'2 - Setup Pattern 2',cr,lf,
'3 - Setup Pattern 3',cr,lf,
'4 - Plot Color Pattern',cr,lf,
'? - Show Polygon Data Structure',cr,lf,cr,lf,
'(Parameter) - Hexadecimal parameter ',
'terminated by a space',cr,lf);

19 1 declare mainprompt (*) byte initial ('Main: ');
20 1 declare turtleprompt (*) byte initial ('Turtle: ');
21 1 declare colorprompt (*) byte initial (cr,lf,'Color Mode');
22 1 declare bwprompt (*) byte initial (cr,lf,'B/W Mode');

/* The POLYGON data structure */
23 1 declare x0(edge limit) address;
24 1 declare y0(edge limit) address;
25 1 declare x1(edge limit) address;
26 1 declare y1(edge limit) address;

/* derived values */
27 1 declare active(edge limit) byte;
/* true if edge is in the active list */
28 1 declare xright(edge limit) byte;
/* true if x1=x2 */
29 1 declare moredx(edge limit) byte;
/* true if abs(slope) is (1/2) */
30 1 declare xint(edge limit) address;
/* the x-intercept of the scanline with this edge */
31 1 declare delx(edge limit) address;
/* absolute value of x2-x1 */
32 1 declare dely(edge limit) address;
/* absolute value of y2-y1 */
33 1 declare rem(edge limit) address;
/* the 'remainder' in the error value */
34 1 declare up(edge limit) address;
/* up/down flag for original edge */

/* two pointers into the data structure */
35 1 declare (polyedges,thisedgeset) address;
/* polyedges = pointer to last valid edge */
/* thisedgeset = pointer to
the start of the current edge set */

```



```

92 2      if (c='a') and (c='f') then d=c-'a'+10;
94 2      else if (c='A') and (c='F') then d=c-'A'+10;
96 2      else if (c='0') and (c='9') then d=c-'0';
98 2      else d=16;
99 2      return(d);
100 2  end code;

101 1  wordin: procedure address;
/* accumulate address parameter
   from console input device */
102 2      declare w address;
103 2      declare (c, char) byte;
104 2      w:=0; c:=0;
106 2      do while c(16);
107 3          w:=shl(w,4)+c; char:=c;
109 3          call co(char); c:=code(char);
111 3      end;
112 2      return(w);
113 2  end wordin;
114 1  setmode: procedure(mode);
/* set the plotting mode */
115 2      declare mode byte;
116 2      if mode/2 then plotmode:=2; else plotmode:=mode;
119 2  end setmode;

120 1  setfont: procedure (font);
/* set the font pattern number used to fill areas */
121 2      declare font byte;
122 2      if font(16) then plotfont:=font;
124 2      else plotfont:=shl(font,3);
125 2  end setfont;

126 1  colorcrt: procedure (boolean);
/* Is this for a Color or a B/W Monitor */
127 2      declare boolean byte;
128 2      colormode:=boolean;
129 2      if colormode then
130 3      do;
131 4          call cstrings(@colorprompt, length(colorprompt));
132 4          screenwidth:=256;
133 4          screenheight:=240;
134 3      end;
135 2      else
136 3      do;
137 4          call cstrings(@bwprompt, length(bwprompt));
138 4          screenwidth:=512;
139 4          screenheight:=480;
140 3      end;
141 2  end colorcrt;

141 1  setcolor: procedure (color);
142 2      declare color byte;
143 2      if (color and 001h) (0) then red:=true; else red:=false;
146 2      if (color and 002h) (0)
147 3      then green:=true; else green:=false;
149 2      if (color and 004h) (0)
150 3      then blue:=true; else blue:=false;
152 2      if (color and 008h) (0)
153 3      then xtra:=true; else xtra:=false;
155 2  end setcolor;

156 1  modify: procedure(wordadr, pattern);
/* modify frame buffer location with pattern */
157 2      declare (wordadr, pattern) address;
158 2      declare (w0, w1, w2, w3) address;
159 2      if colormode then
160 3      do;
161 4          w0:=0; w1:=0; w2:=0; w3:=0;
165 3          if red then
166 4          do;
167 5              w0:=w0 or low(pattern);
168 5              w1:=w1 or high(pattern);
169 4          end;
170 3          if green then
171 4          do;
172 5              w0:=w0 or shl(double(low(pattern)),8);
173 5              w1:=w1 or shl(double(high(pattern)),8);
174 4          end;
175 3          if blue then
176 4          do;
177 5              w2:=w2 or low(pattern); w3:=w3 or high(pattern);
178 4          end;
179 3          if xtra then
180 4          do;
181 5              w2:=w2 or shl(double(low(pattern)),8);
182 5              w3:=w3 or shl(double(high(pattern)),8);
183 4          end;
184 3          wordadr:=shl(wordadr,1);
185 3          do case plotmode;
186 4              do; /* case 0 - merge pattern with buffer */
187 5                  FB0(wordadr)=FB0(wordadr) or w0;
188 5                  FB0(wordadr+1)=FB0(wordadr+1) or w1;
189 5                  FB1(wordadr)=FB1(wordadr) or w2;
190 5                  FB1(wordadr+1)=FB1(wordadr+1) or w3;
191 4              end;
192 4              do; /* case 1 - clear pattern from buffer */
193 5                  FB0(wordadr)=FB0(wordadr) and not w0;
194 5                  FB0(wordadr+1)=FB0(wordadr+1) and not w1;
195 5                  FB1(wordadr)=FB1(wordadr) and not w2;
196 5                  FB1(wordadr+1)=FB1(wordadr+1) and not w3;
197 4              end;
198 4              do; /* case 2 - complement pattern */
199 5                  FB0(wordadr)=FB0(wordadr) xor w0;
200 5                  FB0(wordadr+1)=FB0(wordadr+1) xor w1;
201 5                  FB1(wordadr)=FB1(wordadr) xor w2;
202 5                  FB1(wordadr+1)=FB1(wordadr+1) xor w3;
203 4              end;
204 4          end;
205 3          end;
206 3      else
207 4      do case plotmode;
208 5          /* case 0 - merge pattern with buffer */
209 5          FB0(wordadr)=FB0(wordadr) or pattern;
210 5          /* case 1 - clear pattern from buffer */
211 5          FB0(wordadr)=FB0(wordadr) and not pattern;
212 5          /* case 2 - selectively complement with pattern */
213 5          FB0(wordadr)=FB0(wordadr) xor pattern;
214 4          end;
215 3          end;
216 2      end modify;

213 1  pterror: procedure (x,y) byte;
/* check if the point x,y is in range of the screen */
/* report on the console if error */
214 2      declare (x,y) address;
215 2      if (x)=screenwidth or (y)=screenheight then
216 3      do;
217 4          call cstrings(@pointerror, length(pointerror));
218 4          call wordout(x); call co(','); call wordout(y);
219 4          call crlf;
220 4          return true;
221 3      end;
222 2      else return false;
223 2  end pterror;

226 1  plotpt: procedure (x,y);
/* plot a single pixel at the point x,y */
227 2      declare (x,y) address;
228 2      declare (paddr, px, py) address;
229 2      declare (pmask) address;
230 2      declare (pct) byte;
231 2      if pterror(x,y) then return;
232 2      /* set bit zero */
233 2      pmask:=1;
234 2      pct:=low(x) and 0fh;
235 2      if pct(0) then pmask:=rol(pmask, pct);
236 2      if colormode
237 3      then paddr:=shl(y,5) and 01fe0h or (shl(x,4) and 0fh);
238 3      else paddr:=shl(y,5) and 03fe0h or (shl(x,4) and 01fh);
239 2      call modify(paddr, pmask);
240 2      call plotpt(paddr, pmask);
241 2  end plotpt;

```



```

480 2      call wordout(y1(e)); call co(' ');          /* the lines have the same slope */
482 2      call wordout(deltax(e)); call co(' ');      /* or one of them is horizontal */
484 2      call wordout(dely(e)); call co(' ');      ( (up(p)=up(lastptr))
486 2      call wordout(rem(e)); call crlf;            or ((dely(p)=0) or (dely(lastptr)=0)) )
488 2      end showedge;                               then active(p)=false;

489 1      startpoly: procedure (x,y);
/* start a new polygon data structure */
490 2      declare (x,y) address;
491 2      if pterror(x,y) then return;
/* init the two global indexes into the data structure */
493 2      polyedges=0;
494 2      thisedgeset=0;
/* set the first vertex */
495 2      x0(polyedges)=x;
496 2      y0(polyedges)=y;
497 2      end startpoly;

498 1      addvertex: procedure (x,y);
/* add vertex to the current edge set of the polygon */
499 2      declare (x,y) address;
500 2      if pterror(x,y) then return;
/* complete the second vertex of the last edge */
502 2      x1(polyedges)=x;
503 2      y1(polyedges)=y;
504 2      polyedges=polyedges+1;
/* add the first vertex of the next edge */
505 2      x0(polyedges)=x;
506 2      y0(polyedges)=y;
507 2      end addvertex;

508 1      newedgeset: procedure(x,y);
/* terminate previous edge set and start a new one */
509 2      declare (x,y) address;
510 2      if pterror(x,y) then return;
/* make the last vertex of this edge the same
as the first vertex of this edgeset, thereby
closing the polygon on itself. */
512 2      x1(polyedges)=x0(thisedgeset);
513 2      y1(polyedges)=y0(thisedgeset);
514 2      polyedges=polyedges+1;
515 2      thisedgeset=polyedges;
/* insert the first vertex of the new edgeset */
516 2      x0(polyedges)=x;
517 2      y0(polyedges)=y;
518 2      end newedgeset;

519 1      endpoly: procedure;
/* finish the last edge set of the polygon */
520 2      declare e address;
521 2      x1(polyedges)=x0(thisedgeset);
522 2      y1(polyedges)=y0(thisedgeset);
/* closed off last edge set */
523 2      /* set up/down flag on the accumulated polygon edges */
524 2      do e=0 to polyedges;
525 3          if y1(e)=y0(e) then up(e)=true; else up(e)=false;
526 3      end;
527 2      end endpoly;

528 1      fillpoly: procedure;
/* area fill the polygon specified by data structure */
529 2      declare (x,y,lastptr) address;
530 2      declare (edge,p,ptr,px0,px1,minx,miny,maxy) address;
531 2      nextpt: procedure address;
/* acquire next active x-intercept pt from polygon */
532 3      declare (p,ptr,mx) address;
533 3      ptr=edgelim;
534 3      mx=screenwidth-1;
/* scan polygon edge list for leftmost active pt */
535 3      do p=0 to polyedges;
536 4          if lastptr(edgelim) then
537 5              do;
538 6                  /* discard duplicate points on edges */
539 7                      if (xint(p)=xint(lastptr))
and

```

```

521 7         if xright(edge) then xint(edge)=xint(edge)+1;
523 7         else xint(edge)=xint(edge)-1;
524 7         rem(edge)=rem(edge)-dely(edge);
525 7     end;
526 6     rem(edge)=rem(edge)+delx(edge);
527 6     end;
528 5     else
529 6     /* greater than 45 degree slope */
530 6     do
531 6         rem(edge)=rem(edge)-delx(edge);
532 6         if not(rem(edge)<32768) then
533 7             if xright(edge) then xint(edge)=xint(edge)+1;
534 7             else xint(edge)=xint(edge)-1;
535 7         end;
536 6     end;
537 6     end;
538 5     end;
539 4     end;
540 3     /* Pick up point pairs in ascending order of x */
541 3     /* ptrb and ptrl are indexes into the polygon data */
542 3     /* lastptr is index used in finding duplicate pts */
543 3     ptrb=0; ptrl=0; lastptr=edgelim;
544 4     do while (ptrb<edgelim) and (ptrl<edgelim);
545 4         ptrb=nextpt; pxb=mx;
546 4         ptrl=nextpt; pl=mx;
547 4         /* be sure there are two valid candidates */
548 4         if not((ptrb<edgelim) or (ptrl<edgelim)) then
549 4             call inkline(pxb,pxl,y);
550 4         end;
551 3     end;
552 2     end fillpoly;
553 1     linepoly: procedure;
554 2     /* draw edge segments of polygon as line segments */
555 2     declare e address;
556 3     do e=0 to polyedges;
557 3         call dda(x0(e),y0(e),x1(e),y1(e));
558 2     end;
559 1     showpoly: procedure;
560 2     /* print the data structure on the console */
561 2     declare e address;
562 2     call crlf;
563 2     call cstrings(edgetitle,length(edgetitle));
564 3     do e=0 to polyedges;
565 3         call showedge(e);
566 2     end;
567 1     fantasy: procedure(xs,ys);
568 2     /* line drawing exerciser with interesting patterns */
569 2     declare (xs,ys) address;
570 2     declare (x,y) address;
571 2     if xs<screenwidth-1 then xs=screenwidth-1;
572 2     if ys<screenheight-1 then ys=screenheight-1;
573 2     do x=0 to screenwidth-1;
574 2         if colormode then call setcolor(x and 0fh);
575 3         call dda(xs,ys,x,screenheight-1);
576 3         if csts then do call co(ci); return; end;
577 3     end;
578 2     do y=0 to screenheight-1;
579 3         if colormode then call setcolor(y and 0fh);
580 3         call dda(xs,ys,screenwidth-1,screenheight-y-1);
581 3         if csts then do call co(ci); return; end;
582 3     end;
583 2     do x=0 to screenwidth-1;
584 3         if colormode then call setcolor(x and 0fh);
585 3         call dda(xs,ys,screenwidth-x-1,0);
586 3         if csts then do call co(ci); return; end;
587 3     end;
588 2     do y=0 to screenheight-1;
589 3         if colormode then call setcolor(y and 0fh);
590 3         call dda(xs,ys,screenwidth-x-1,0);
591 3         if csts then do call co(ci); return; end;
592 3     end;
593 2     end;
594 2     do y=0 to screenheight-1;
595 3         if colormode then call setcolor(y and 0fh);
596 3         call dda(xs,ys,screenwidth-x-1,0);
597 3         if csts then do call co(ci); return; end;
598 3     end;
599 2     end;
600 2     do y=0 to screenheight-1;
601 3         if colormode then call setcolor(y and 0fh);
602 3         call dda(xs,ys,screenwidth-x-1,0);
603 3         if csts then do call co(ci); return; end;
604 3     end;
605 3     if colormode then call setcolor(y and 0fh);
606 3     call dda(xs,ys,0,y);
607 3     if csts then do call co(ci); return; end;
608 3     end;
609 2     end;
610 1     clearscreen: procedure(val);
611 2     declare val address;
612 2     /* take advantage of the 8085 string operations */
613 2     call seth(val,0fb0,length(0fb0));
614 2     call seth(val,0fb1,length(0fb1));
615 2     end clearscreen;
616 1     turtle: procedure;
617 2     /* the turtle is a simple graphic input device. */
618 2     /* turtle shows itself as a rectangle on the CRT. */
619 2     /* the user may make the turtle crawl */
620 2     /* up, down, right, and left. */
621 2     /* the crawl speed of the turtle is adjustable. */
622 2     /* using current position of turtle, several commands */
623 2     /* may be invoked which contribute vertices to */
624 2     /* polygon data structure for later plotting. */
625 2     /* vertex commands are: startpolygon, addvertex, */
626 2     /* newedgeset, and endpolygon. */
627 2     declare char brite;
628 2     declare (x,y,delta) address;
629 2     declare (oldmode,oldfont,inking) bytes;
630 2     toggleturtle: procedure(x,y);
631 2     /* since the plotmode has been set to 2 */
632 2     /* this procedure complements rectangular turtle */
633 2     declare (x,y) address;
634 2     call plotrect(x-2,y-2,x+2,y+2);
635 2     end toggleturtle;
636 2     crawlto: procedure(xnew,ynew);
637 2     /* crawl to new coordinate from present position. */
638 2     /* first erase line segment to last entered vertex */
639 2     /* then draw line to new turtle position and */
640 2     /* add the turtle. */
641 2     declare (xnew,ynew) address;
642 2     call toggleturtle(x,y);
643 2     if inking then
644 3     do
645 3         call dda(x0(polyedges),y0(polyedges),x,y);
646 3         call dda(x0(polyedges),y0(polyedges),xnew,ynew);
647 3     end;
648 2     x=xnew; y=ynew;
649 2     call toggleturtle(x,y);
650 2     end crawlto;
651 2     oldmode=plotmode; oldfont=plotfont;
652 2     inking=false;
653 2     plotmode=2; plotfont=0; delta=4;
654 2     char=0;
655 2     x=screenwidth/2; y=screenheight/2;
656 2     /* initial turtle position */
657 2     call toggleturtle(x,y);
658 2     do while (char(0)'E');
659 3     call crlf;
660 3     call cstrings(turtleprompt,length(turtleprompt));
661 3     char=ci;
662 3     call co(char);
663 3     char=upcase(char);
664 3     if char='C' then delta=wordin; /* set crawl speed */
665 3     if char='U' then call crawlto(x,y-delta); /* up */
666 3     if char='D' then call crawlto(x,y+delta); /* down */
667 3     if char='L' then call crawlto(x-delta,y); /* left */
668 3     if char='R' then call crawlto(x+delta,y); /* right */
669 3     if char='S' then /* start polygon */
670 3     do
671 3         call startpoly(x,y);
672 3         inking=true;
673 3     end;
674 3     if char='A' then /* add vertex to current edgeset */
675 3     do

```

```

673 4      call addvertex(x,y);
674 4      end;
675 3      if char='E' then /* end polygon and exit turtle */
676 3      do;
677 4          call endpoly;
678 4          call dda(x0(polyedges),y0(polyedges),
              x1(polyedges),y1(polyedges));
679 4      end;
680 3      if char='N' then /* begin new edgset */
681 3      do;
        /* add line to close off last edgset */
682 4          call dda(x0(polyedges),y0(polyedges),
              x0(thisedgset),y0(thisedgset));
        /* erase the turtles latest line segment */
683 4          call dda(x,y,x0(polyedges),y0(polyedges));
684 4          call newedgset(x,y);
685 4      end;
686 3      if char='H'
        then call cstrings(4turtlehelp,length(turtlehelp));
688 3      end;
689 2      call togsieturtle(x,y);
690 2      plotmode=oldmode; plotfont=oldfont;
692 2      end turtle;
        /* Here are three simple polygons to demonstrate
        how fixed shapes might be programmed */
693 1      setup1: procedure;
        /* An engineering symbol */
694 2      declare s address;
695 2      if colormode then s=1; else s=2;
696 2      call startpoly(25*s,50*s);
697 2      call addvertex(50*s,25*s);
700 2      call addvertex(50*s,40*s);
701 2      call addvertex(100*s,40*s);
702 2      call addvertex(100*s,25*s);
703 2      call addvertex(125*s,50*s);
704 2      call addvertex(100*s,75*s);
705 2      call addvertex(100*s,60*s);
706 2      call addvertex(50*s,60*s);
707 2      call addvertex(50*s,75*s);
708 2      call endpoly;
709 2      end setup1;
710 1      setup2: procedure;
        /* The numbers 8 and 6 from 8886 */
711 2      declare s address;
712 2      if colormode then s=1; else s=2;
        /* The outside edge of the 8 */
715 2      call startpoly(50*s,100*s);
716 2      call addvertex(50*s,200*s);
717 2      call addvertex(100*s,200*s);
718 2      call addvertex(100*s,100*s);
        /* One hole in the 8 */
719 2      call newedgset(60*s,110*s);
720 2      call addvertex(60*s,145*s);
721 2      call addvertex(90*s,145*s);
722 2      call addvertex(90*s,110*s);
        /* A second hole in the 8 */
723 2      call newedgset(60*s,155*s);
724 2      call addvertex(90*s,155*s);
725 2      call addvertex(60*s,190*s);
726 2      call addvertex(60*s,190*s);
        /* The outline of the 6 */
727 2      call newedgset(125*s,100*s);
728 2      call addvertex(175*s,100*s);
729 2      call addvertex(175*s,110*s);
730 2      call addvertex(135*s,110*s);
731 2      call addvertex(135*s,145*s);
732 2      call addvertex(175*s,145*s);
733 2      call addvertex(175*s,200*s);
734 2      call addvertex(125*s,200*s);
        /* The hole in the 6 */
735 2      call newedgset(135*s,155*s);
736 2      call addvertex(165*s,155*s);
737 2      call addvertex(165*s,190*s);
738 2      call addvertex(135*s,190*s);
739 2      call endpoly;
740 2      end setup2;
741 1      setup3: procedure;
        /* A typical polygon */
742 2      declare s address;
743 2      if colormode then s=1; else s=2;
744 2      call startpoly(150*s,50*s);
747 2      call addvertex(200*s,75*s);
748 2      call addvertex(225*s,150*s);
749 2      call addvertex(250*s,15*s);
750 2      call addvertex(225*s,50*s);
751 2      call addvertex(200*s,25*s);
752 2      call endpoly;
753 2      end setup3;
754 1      setup4: procedure;
        /* a color test */
755 2      declare (x,y) address;
756 2      do y=0 to 7;
757 3      call setcolor(y);
758 3      call plotrect(0,y*30,239,(y+1)*30-1);
759 3      end;
760 2      do x=0 to 7;
761 3      call setcolor(x);
762 3      call plotrect(x*30,0,(x+1)*30-1,239);
763 3      end;
764 2      end setup4;
        /* the main program loop */
        /* after clearing the frame buffer */
        /* and establishing the initial modes */
        /* it simply tests the console input characters */
        /* for a variety of user selected commands */
        /* At either Main or Turtle command level */
        /* User may type the character "h" for help */
765 1      call cstrings(81main,length(81main));
766 1      call clearscreen(00000h);
767 1      ;
768 1      call setmode(2);
769 1      call setfont(0);
770 1      call setcolor(0fh);
        /* red, green, blue, and xtra yields white */
771 1      call colorcrt(true);
772 1      call crif;
773 1      polyedges=0;
774 1      do while true; /* do forever */
775 2      call cstrings(81mainprompt,length(81mainprompt));
776 2      char=c;
777 2      call co(char);
778 2      char=upcase(char);
779 2      if (char='M') then call setmode(wordin);
781 2      if (char='B') then call setcolor(wordin);
783 2      if (char='O') then call colorcrt(not colormode);
785 2      if (char='I') then call setfont(wordin);
787 2      if (char='S') then call startpoly(wordin,wordin);
789 2      if (char='A') then call addvertex(wordin,wordin);
791 2      if (char='E') then call endpoly;
793 2      if (char='N') then call newedgset(wordin,wordin);
795 2      if (char='P') then call fillpoly;
797 2      if (char='C') then call clearscreen(00000h);
799 2      if (char='F') then call fantasy(wordin,wordin);
801 2      if (char='H') then call cstrings(81mainhelp,length(81mainhelp));
803 2      if (char='L') then call linepoly;
805 2      if (char='T') then call turtle;
807 2      if (char='1') then call setup1;
809 2      if (char='2') then call setup2;
811 2      if (char='3') then call setup3;
813 2      if (char='4') then call setup4;
815 2      if (char='?') then call showpoly;
817 2      call crif;
818 2      end;
819 1      end pixel;

```



APPENDIX E

MODULE INFORMATION:

CODE AREA SIZE = 1F83H 8867D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 06AFH 1711D

MAXIMUM STACK SIZE = 0046H 70D
1064 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION

INA 950-1 LOCAL AREA NETWORK SOFTWARE

- Provides Reliable "Virtual Circuit" Process-to-Process Message Delivery Service
- Pre-standard ISO OSI Transport Control Layer Services:
 - Guaranteed Message Integrity
 - Data Rate Matching (Flow Control)
 - Multiple Connection Capability (Process Multiplexing)
 - Variable-Length Message Support
- Seven Transport Commands Are Available to Client S/W
- iNA Services Are Operating System Independent via MIP in Multibus Applications
- Provides Comprehensive Network Management Functions to Enable Maintenance and Efficient Operation
- Pre-standard ISO OSI Network Management Services:
 - Collection of Network Usage Statistics
 - Transport and Data Link Parameter Inspection and Setting Capability
 - Fault Detection and Isolation
 - Initialization Support
- Seven Network Management Commands Available to Client S/W
- iNA Implemented on Intel's iSBC® 550 Ethernet* Controller Board Set

iNA 950-1 is a RAM-based software which provides a high-level reliable message transport service plus extensive network management functions to help maintain and efficiently operate a local area network.

iNA is implemented on Intel's iSBC 550 Ethernet Communications Controller Board set and when connected to an Ethernet (Version 1.0, Sept. '80) transceiver or equivalent (like INTELLINK™), it forms a complete Ethernet local area network communications subsystem for Multibus®-based network system applications. iNA 950-1 together with the iSBC 550 implements the physical and data link layers (Ethernet) and the transport ("class 4") layer functions of the ISO OSI Model.

iNA 950-1 is the first software product in Intel's family of LAN building blocks. In addition, it is the first product toward fully implementing a family of ISO OSI standard S/W products.

*Ethernet is a trademark of Xerox Corporation.

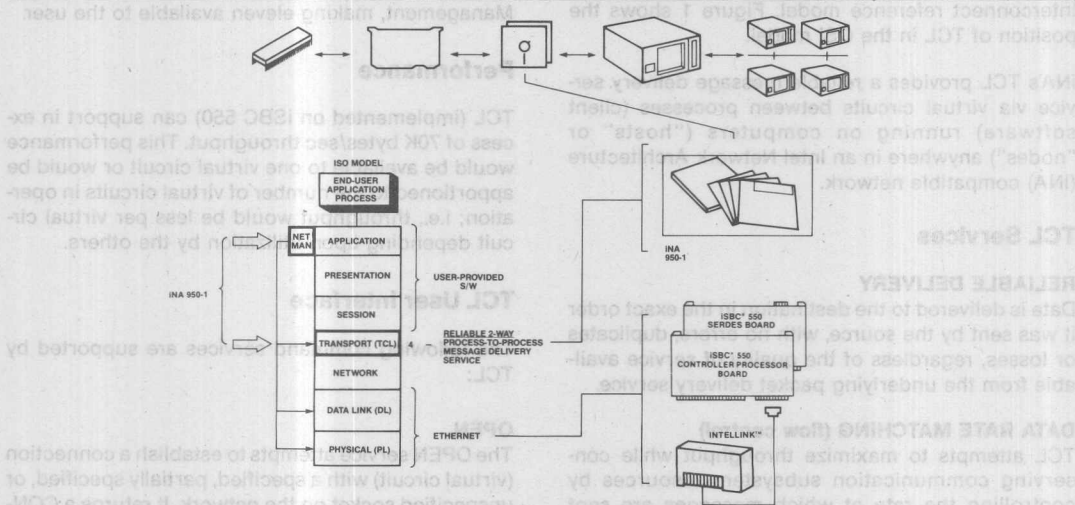


Figure 1. Intel's Ethernet* LAN Building Block Family

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

©INTEL CORPORATION, 1982

INA 950-1 FUNCTIONAL DESCRIPTION

INA 950-1 is a RAM-based software which provides reliable process-to-process communication for implementation on Intel's iSBC 550 Ethernet Communications Controller board set in Multibus-based applications.

The INA 950-1 design is a pre-standard implementation of the Transport "class 4" Control Layer of the ISO OSI model. The Transport Control Layer (TCL) ensures a reliable, full-duplex-message delivery service on top of the "best effort" Ethernet packet service provided by the iSBC 550 physical/data link functions.

The INA 950-1 design also includes a Network Management function (NM), which provides services to help support and effectively maintain the operation of the network. It is the residence of tools and offline utilities which perform such services as booting up new nodes; performing error recognition and logging to aid in problem isolation and resolution; and supporting network planning by maintaining statistics on network loading and utilization.

The client OS (RMX-86, RMX-88, and others) can easily interface to INA via MMX 800 (MIP).

TCL (TRANSPORT CONTROL LAYER) FUNCTIONAL DESCRIPTION

INA 950-1 is an implementation of the Intel Network Architecture, which follows the ISO Open Systems Interconnect reference model. Figure 1 shows the position of TCL in the OSI model.

INA's TCL provides a reliable message delivery service via virtual circuits between processes (client software) running on computers ("hosts" or "nodes") anywhere in an Intel Network Architecture (iNA) compatible network.

TCL Services

RELIABLE DELIVERY

Data is delivered to the destination in the exact order it was sent by the source, with no errors, duplicates or losses, regardless of the quality of service available from the underlying packet delivery service.

DATA RATE MATCHING (flow control)

TCL attempts to maximize throughput while conserving communication subsystem resources by controlling the rate at which messages are sent based on its own resources and the availability of receive buffers at the destination.

PROCESS MULTIPLEXING

Several processes can simultaneously use TCL with no risk of interference with others.

VARIABLE-LENGTH MESSAGES

The client software can submit arbitrarily short or long messages for transmittal without regard for the minimum or maximum packet lengths supported by the underlying packet delivery service.

Virtual Circuits

TCL provides these services, with a connection (or virtual circuit) service: Pairs of clients create, send data over, and terminate connections between themselves.

Client processes are identified by means of sockets. A socket consists of a Network ID, a Host ID, and a Port. It is the network-wide address of a process (or related group of processes) running on a host in the network.

The Port specifies the process within a host, the Host ID specifies which physical node in the network, and the Network ID specifies a portion of a large inter-networked system. Host IDs are unique over all hosts ever installed over time, so net IDs are actually redundant information that provides additional routing information to the Network (routing) layer, when present.

TCL is capable of supporting up to twelve virtual circuits simultaneously. One is dedicated to Network Management, making eleven available to the user.

Performance

TCL (implemented on iSBC 550) can support in excess of 70K bytes/sec throughput. This performance would be available to one virtual circuit or would be apportioned to the number of virtual circuits in operation; i.e., throughput would be less per virtual circuit depending upon utilization by the others.

TCL User Interface

The following command services are supported by TCL:

OPEN

The OPEN service attempts to establish a connection (virtual circuit) with a specified, partially specified, or unspecified socket on the network. It returns a CONNECTION ID which is used by the client to refer to this virtual circuit in other TCL service requests.

SEND

The SEND service queues the client's transmit buffer on the virtual circuit specified by the CONNECTION ID. The messages will be transmitted in the order they are queued on each virtual circuit. The transmit buffer can consist of noncontiguous areas of memory blocks.

POST RECEIVER BUFFER

Clients use POST RECEIVER BUFFER service to supply the empty buffer to TCL to place messages sent to this socket. The TCL queues the receiver buffer on the virtual circuit specified by the CONNECTION ID. The receiver buffers may be noncontiguous.

STATUS

The STATUS service places the connection-independent information (such as ... number of virtual circuits established), and the connection-dependent information (such as ... number of packets transmitted to a particular CONNECTION ID) into the client specified buffer.

DEFERRED STATUS

DEFERRED STATUS instructs TCL to notify the client when the connection enters the established state. This service provides the client with the recognition of connection establishment without using successive interrogation of the connection state via the STATUS request.

CLOSE

A CLOSE request initiates the graceful termination of the virtual circuit specified by the CONNECTION ID. TCL will complete the transmission of this virtual circuit's previously queued local transmit buffer before closing.

ABORT

The ABORT service immediately terminates the virtual circuit specified by the CONNECTION ID, releases the CONNECTION ID, and returns any other locally outstanding request blocks.

NM (NETWORK MANAGEMENT) FUNCTIONAL DESCRIPTION

Network Management provides functions to aid in the planning, operation, and maintenance of a multi-station network. Planning functions include collection of network usage statistics; operation functions include parameter inspection and setting capability; and maintenance functions include access to layer statistics and loopback tests.

Network Management-Accessible Objects

Each layer allows Network Management access to selected parts of its internal data base (i.e., "objects"). Normally objects are read-only quantities. There are two special types of objects: parameters and counters, which may be modified by the users.

- Parameters are values which adjust the operational characteristics of a layer. They may be adjusted by using the SET operation.
- Counters record the number of times some event occurs. They may be read, or read/then cleared.

NM Services**COLLECTION OF NETWORK USAGE STATISTICS**

Thirty-six individual network values ("objects"), twenty-five from transport layer and eleven from data link layer (implemented by iSBC 550), are accessible to the client software via the Network Management functions. Within the TCL, fourteen connection-independent objects (i.e., number of virtual circuits open) are available and eleven connection-dependent objects (i.e., number of packets sent for a particular connection ID) are also available.

Examples of objects available from data link layer are: number of collisions at a node, or the number of packets discarded due to CRC errors. This information may be obtained locally, or remotely from other nodes, making possible network management from a single dedicated node.

Additionally, maintenance services are provided by Network Management's capability of performing loopback tests to any other node on the network. For example, packets can be echoed off of other nodes which essentially tests that node's receiving and transmitting functions.

PARAMETER INSPECTION AND SETTING CAPABILITY

Network Management enables the client to inspect such TCL parameters as timeouts and make changes to help improve node efficiency based on network usage statistics.

INITIALIZATION SUPPORT

Network Management assists in initializing and booting up nodes both locally and remotely ("down-line loading").

iNA Network Management, coupled with data link diagnostics resident in the iSBC 550, form a comprehensive network maintenance and planning resource critical to the success of networked systems in end-user environments.

NM User Interface

The following seven command services are supported by NM:

READ

Returns the value of the specified objects to the client.

READC

Reads and if any of the objects are counters, they are cleared to zero.

SET

Assigns a new value to the specified objects.

CAUSE EVENT

Allows users to add connection data base memory to the Transport Control Layer.

READ MEMORY AND SET MEMORY

This interface is provided as a debugging feature. It can be used to read or set the counters of any part of memory as seen by the iSBC 550, local or remote.

NM Bootstrap Server Interface

The following two interfaces support the bootstrap server to directly access the Data Link:

SUPPLY BUF

This service is used to provide NM with buffers in which to place received packets.

TAKE BACK

Causes NM to return all the receiver buffer to the client.

iSBC 550 IMPLEMENTATION/INTERFACE DETAILS

iNA 950-1 is implemented as RAM-based software on the processor board of the iSBC 550 Communications Controller board set. iNA 950-1 and the iSBC 550 firmware operate under control of a resident realtime executive running on the 8088 CPU. The firmware consists of the Multibus Interprocessor Protocol (MIP), those data link functions not supported by hardware, a bootstrap loader, and diagnostics. Together they provide the Transport, Data Link and the Physical Link functions of the ISO OSI model, in addition to the Network Management function. (See Fig. 2.)

The iNA-iSBC 550 communications subsystem is controlled from the Multibus host CPU via the Multibus Interprocessor Protocol (MIP). Upon system start-up or reset, the client software initiates the loading of iNA 950-1 object-code into the iSBC 550 RAM. This is performed by the bootstrap loader resident in the firmware. The loading can be either locally from the host or remotely via the network and the code may be located either locally or remotely. iNA 950-1 requires approximately 15K bytes of RAM memory.

In order to issue a request to the communications processor, the host generates a request-block and uses the iMMX 800 Transfer Message Interface to send the request-block to the MIP socket on the iSBC 550. iMMX 800 is an implementation of MIP for iRMX™ 80, 86, and 88. The request-block consists of specifying the command (OPEN, SEND, etc.) and the values of the parameters required. TCL/NM perform their functions on top of the services provided by the data/physical link functions of the iSBC 550 hardware. Upon completion of the requested service, TCL/NM return the request-block, containing the response, to the host via MIP.

ORDERING INFORMATION

Part Number	Description
iNA 950-1	<ul style="list-style-type: none"> • Seven Intellect®-compatible diskettes (written in PL/M assembly language) containing <ul style="list-style-type: none"> —Source code and listings —Object code —Source-to-object generation tools —Application examples

- Programmer's Reference Manual
- Architecture Reference Manual

iDCM 911-1 INTELLINK™ ETHERNET* CLUSTER MODULE

- Eliminates need for transceivers and Ethernet coaxial cable for a local cluster of workstations
- Enables local cluster of nine workstations to connect to main Ethernet cable with only one transceiver
- Enables workstations to be up to 100M from main Ethernet* cable
- Complies with the Ethernet Specification, Version 1.0, September 1980

The Intellink™ Ethernet Cluster Module is a device used as a means of interconnecting up to nine Ethernet devices without the need for Ethernet coaxial cable and transceivers. The Intellink module forms a standalone Ethernet local area network with "interconnection" communication capability. The Intellink module (and attached devices) can optionally be connected to the Ethernet coaxial cable through a single transceiver.

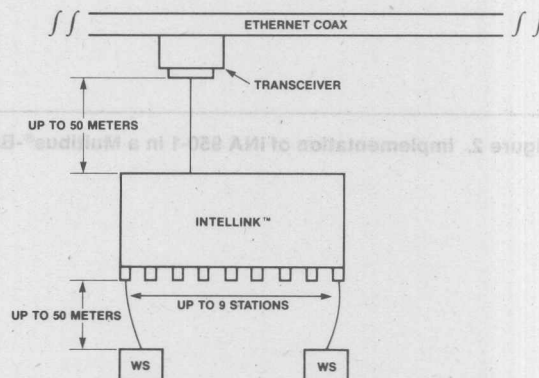
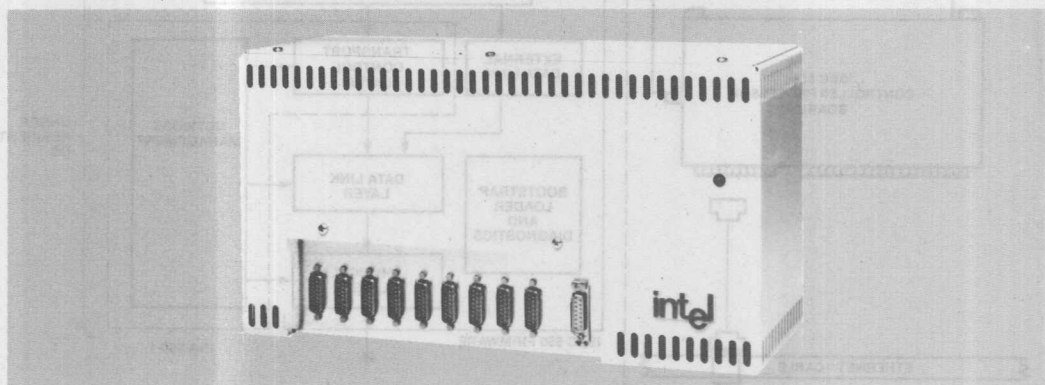


Figure 1. Intellink™ Configuration

*Ethernet is a trademark of Xerox Corporation.

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied. Information Contained Herein Supersedes Previously Published Specifications On These Devices From Intel.

FUNCTIONAL DESCRIPTION

Intellink module performs the same functions as a standard Ethernet transceiver. It buffers receive and transmit data, detects attempts by two or more stations to gain access to the line simultaneously, signals the presence of a collision to the transmitting stations, and transmits the jam signal prior to initiation of the random back-off algorithm. It complies with all of the interface parameters set forth in "The Ethernet Specification," 1.0 Version, September 1980.

Ethernet Work Station to Intellink™ Interface (WI) Connectors

There are nine WI interface connectors into which Ethernet-based systems can be connected. Each connector has the same signal pairs as does the equivalent connector on a standard Ethernet transceiver.

Intellink™ Module to Transceiver Interface (IT) Connector

The IT interface connector on the Intellink module is used to connect the local cluster to the "main" Ethernet cable through a standard transceiver, or can be left unconnected for standalone operation. The characteristics of this connector are identical to an Ethernet system to transceiver cable connector.

Topology

The Intellink module can function in standalone operation in which case it appears as a "zero length Ethernet segment" for up to nine Ethernet-based

systems, or optionally can be connected to the "main" Ethernet coaxial cable through a single transceiver. When connected to the "main" Ethernet coaxial cable, it extends the Ethernet system interface to the transceiver from 50 meters to 100 meters. (Figure 1).

Physical Characteristics

Width	14 in. (35.56 cm)
Height	7.8 in. (19.81 cm)
Depth	5.5 in. (13.97 cm)
Weight	10 lb. (4.52 kg)

ELECTRICAL CHARACTERISTICS

Input Voltage Range: (Voltages AC RMS)

Voltage (15%)
100V ±15%
120V ±15%
220V ±15%
240V ±15%

NOTE: The frequency range is 47 to 64 Hz, single phase.

ENVIRONMENTAL CHARACTERISTICS

Temperature:	10° to 40°C Operating -40° to 70°C Non-Operating
Humidity:	10% to 85% Operating 5% to 95% Non-Operating

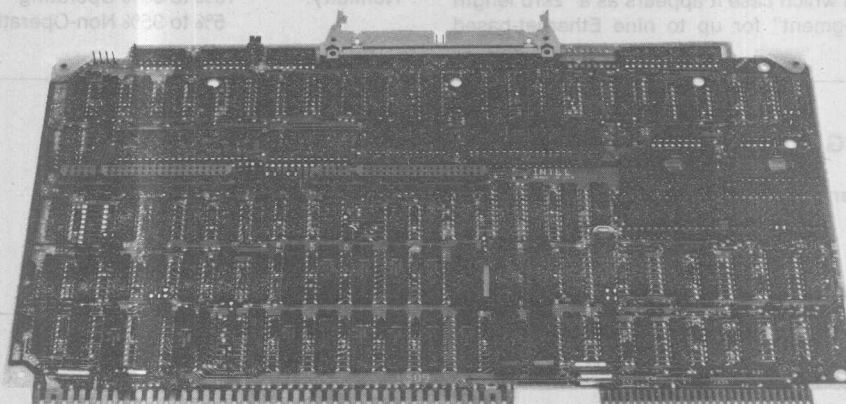
ORDERING INFORMATION

Part Number	Description
IDCM 911-1	Intellink, Ethernet cluster module, Version 1.0

iSBC® 589 INTELLIGENT DMA CONTROLLER

- Configurable as either an intelligent slave or MULTIBUS® master
- 5 MHz 8089 I/O Processor
- MULTICHANNEL DMA I/O bus interface with Supervisor, Controller or Basic Talker/Listener capabilities
- Two 8/16-bit iSBX™ bus connectors
- DMA transfer rates up to 1.25 megabytes per second
- User Command Interface Firmware Package provides high level I/O commands
- 8K bytes of high-speed dual-ported static read/write memory
- Sockets for up to 32K bytes of read only memory or additional byte-wide static RAMs
- Three programmable timers

The iSBC 589 Intelligent DMA Controller is a member of Intel's complete line of MULTIBUS microcomputer systems which take full advantage of VLSI technology to provide economical computer based solutions for OEM applications. The iSBC 589 board is a general purpose, programmable, high-speed DMA controller on a single 6.75 x 12.00 inch printed circuit board. Using the board's dual-port RAM and standard EPROM resident firmware, the on-board Intel 8089 I/O Processor can perform memory to memory block transfers and complex I/O operations via two iSBX connectors and the MULTICHANNEL I/O bus at DMA transfer rates up to 1.25 megabytes per second. Acting as an intelligent slave to one or more iSBC 86, iSBC 88 or iSBC 80 single board computers, the iSBC 589 board enhances the system's overall performance by relieving the host CPU of time consuming I/O operations. The board's unique combination of performance, on-board intelligence and flexible hardware I/O interfaces make the iSBC 589 board the ideal solution for applications with specialized I/O requirements, such as high-speed data acquisition, graphics, instrument automation and specialized peripheral control, that previously would have necessitated an expensive custom designed I/O controller.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Two Modes of Operation

The iSBC 589 Intelligent DMA Controller is capable of operating either as a stand-alone, high-speed data acquisition controller or as an intelligent slave. In stand-alone mode, external requests cause the Intel 8089 I/O Processor to execute I/O programs contained in its on-board memory. As an intelligent slave to one or more Intel single board computers, the IOP can perform sophisticated DMA operations in response to high level commands issued by the host processor. While operating in either mode, the iSBC 589 board may act as a MULTIBUS master to access any system memory or I/O resources.

Input/Output Processor

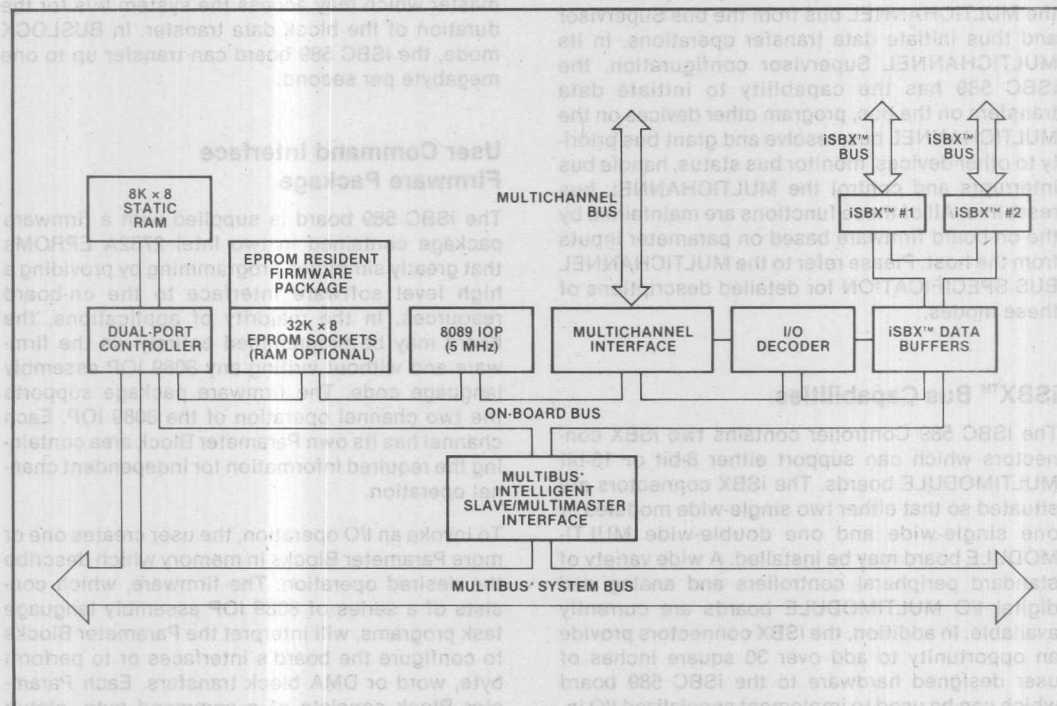
The iSBC 589 board contains a 5 MHz Intel 8089 HMOS I/O Processor, whose architecture and instruction set have been optimized for performing DMA operations. The DMA function of the 8089 IOP uses a two cycle approach where the information actually flows through the 8089 IOP. This ap-

proach to DMA vastly simplifies the bus timings and enhances compatibility with memory and peripherals, in addition to allowing operations to be performed on the data as it is transferred. Operations can include such constructs as translate, where the 8089 automatically vectors through a lookup table and mask compare, both on the "fly". This DMA capability includes flexible termination conditions (such as external terminate, mask compare, single transfer and byte count expired).

The 8089 IOP supports two logically and physically separate I/O channels. The IOP maintains separate register sets for each I/O channel which allows the processor to alternate operation between the two channels without incurring context switching overhead delays.

DMA Capabilities

The iSBC 589 board supports both individual byte or word data transfers and DMA block transfer operations among its MULTICHANNEL interface, two iSBX connectors, on-board RAM and the MULTIBUS interface. Each of these devices may be combined



with any other as the source and destination for a DMA operation. The same firmware commands are used for all of the DMA source and destination combinations.

MULTICHANNEL Capabilities

The MULTICHANNEL bus provides a high-speed 8-bit or 16-bit wide data path for block data transfers between external devices, such as instruments, peripherals and other computers, and the iSBC 589 board. The iSBC 589 board can access up to 15 other devices on the MULTICHANNEL bus at distances of up to 15 meters and has the ability to address up to 16 megabytes of memory and 16 megabytes of I/O on each device.

The iSBC 589 Intelligent DMA Controller can interface to the MULTICHANNEL bus in one of three modes: as a Basic Talker/Listener, a Controller, and a Supervisor. In Basic Talker/Listener Mode, the iSBC 589 board monitors the MULTICHANNEL for requests from a Controller or the bus Supervisor to perform a read or a write operation, but it has no bus control capabilities. In Controller Mode, the board can request temporary control of the MULTICHANNEL bus from the bus Supervisor and thus initiate data transfer operations. In its MULTICHANNEL Supervisor configuration, the iSBC 589 has the capability to initiate data transfers on the bus, program other devices on the MULTICHANNEL bus, resolve and grant bus priority to other devices, monitor bus status, handle bus interrupts and control the MULTICHANNEL bus reset line. All of these functions are maintained by the on-board firmware based on parameter inputs from the host. Please refer to the MULTICHANNEL BUS SPECIFICATION for detailed descriptions of these modes.

iSBX™ Bus Capabilities

The iSBC 589 Controller contains two iSBX connectors which can support either 8-bit or 16-bit MULTIMODULE boards. The iSBX connectors are situated so that either two single-wide modules or one single-wide and one double-wide MULTIMODULE board may be installed. A wide variety of standard peripheral controllers and analog and digital I/O MULTIMODULE boards are currently available. In addition, the iSBX connectors provide an opportunity to add over 30 square inches of user designed hardware to the iSBC 589 board which can be used to implement specialized I/O interfaces. For more information on specific iSBX

MULTIMODULE boards, consult the Intel OEM Microcomputer System Configuration Guide.

MULTIBUS® Capabilities

MULTIBUS system memory and I/O resources may be used as the source or the destination for an iSBC 589 board transfer operation. The iSBC 589 DMA Controller may also be used as a high-speed data mover to transfer blocks of data from one MULTIBUS system RAM area to another. MULTIBUS system memory may also be used to store Parameter Blocks to be executed by the on-board firmware package. The iSBC 589 board, acting as a MULTIBUS Master, can access up to 16 megabytes of MULTIBUS memory and up to 64K MULTIBUS I/O locations.

Two MULTIBUS transfer modes are available. Selection of the desired mode is done via the Parameter Block. Transfer rates of up to 900K bytes per second may be achieved in shared bus mode, where the iSBC 589 board requests access to the system bus for 1.4 microseconds to transfer one byte or word to or from memory. In BUSLOCK mode, the iSBC 589 is established as the sole master which may access the system bus for the duration of the block data transfer. In BUSLOCK mode, the iSBC 589 board can transfer up to one megabyte per second.

User Command Interface Firmware Package

The iSBC 589 board is supplied with a firmware package contained in two Intel 2732A EPROMs that greatly simplifies programming by providing a high level software interface to the on-board resources. In the majority of applications, the board may be programmed entirely via the firmware and without writing any 8089 IOP assembly language code. The firmware package supports the two channel operation of the 8089 IOP. Each channel has its own Parameter Block area containing the required information for independent channel operation.

To invoke an I/O operation, the user creates one or more Parameter Blocks in memory which describe the desired operation. The firmware, which consists of a series of 8089 IOP assembly language task programs, will interpret the Parameter Blocks to configure the board's interfaces or to perform byte, word or DMA block transfers. Each Parameter Block consists of a command byte, status byte, data source and destination pointers and

other information as shown in Table 1. Commands recognized by the firmware package are listed in Table 2. The Execute User Task command is of special interest because it allows the user to extend the capabilities of the iSBC 589 board by adding his own 8089 IOP assembly language routines to the firmware package, while retaining the structure and standard functions supplied by the firmware.

Table 1. User Command Interface Firmware Parameter Block Byte Format

Command Byte
Status Byte
Command Chaining Pointer
Command Chaining Pointer
Command Chaining Pointer
Device Number
MULTICHANNEL Data Type
Memory Pointer or Register Number
Memory Pointer or Register Number
Memory Pointer or Data Storage Location
Memory Pointer or Data Storage Location
Device Number
MULTICHANNEL Data Type
Memory Pointer or Register Number
Memory Pointer or Register Number
Memory Pointer
Memory Pointer
Byte Counter
Byte Counter
Byte Counter

In addition to executing transfer operations, the firmware package executes an initialization sequence which prepares the 8089 IOP and the on-board RAM, EPROM and I/O resources for further firmware execution.

RAM Capabilities

In its standard configuration, the iSBC 589 board contains 8K bytes of high-speed, dual-ported static RAM. The first 256 bytes are dedicated for use by the on-board firmware. The remaining on-board RAM may be used for storing additional Parameter Blocks for the firmware or as a data buffer for I/O operations. This memory is always addressed by the 8089 IOP as locations 0000H to 1FFFFH. However, for MULTIBUS accesses through the dual-port, the RAM base address may be configured on any 8K-byte boundary in the first megabyte page of the MULTIBUS memory space. Users may install additional on-board RAM by placing two byte-wide RAMs in the 28-pin JEDEC standard sockets. The additional RAM is accessible only by the on-board 8089 IOP.

EPROM Capabilities

The iSBC 589 board can be configured with up to 32K bytes of non-volatile read only memory. Four 28-pin sockets are provided for the use of Intel 2716, 2732 and 2764 EPROMs or byte-wide RAMs.

Table 2. User Command Interface Firmware Package Commands

Command	Description
NO-OP	Test the intelligent slave interface on the iSBC 589 board. The board reads the Parameter Block, generates status and interrupts the host on completion.
REGISTER WRITE	Write either a word or byte of data from the Data Storage Location within the Parameter Block to the location specified by the Parameter Block Device Number and Register Number.
REGISTER READ	Read either a word or byte of data from the location specified by the Parameter Block Device Number and Register Number to the Data Storage Location within the Parameter Block.
PERFORM DMA	Transfer data beginning at the location specified by the source Memory Pointer, Device Number and Register Number parameters to the location specified by the destination Memory Pointer, Device Number and Register Number parameters. The number of transfers is specified by the Byte Count parameter. A Byte Count of 0 enables DMA until an external terminate condition is sensed.
EXECUTE USER TASK	Transfer 8089 IOP program execution from the Firmware Package to a user defined 8089 assembly language routine beginning at the location specified by the Memory Pointer parameter. Upon completion, the user task returns control to the firmware.

In the default configuration, the board is jumpered for 32K devices, and, two 2732A EPROMs containing the firmware package are installed. Users who wish to extend the capabilities of the firmware may do so by programming unused locations in the firmware PROMs, installing two additional 2732A PROMs or copying the firmware into 2764s along with their own code. As an alternative, two byte-wide RAMs of equal or smaller capacity may be installed in the open sockets and used in conjunction with the firmware PROMs.

Programmable Interval Timers

Three independent, fully programmable 16-bit interval/event counters are provided by an 8254-12 Programmable Interrupt Timer. Each counter may operate in either BCD or binary mode. One counter is used by the firmware package, leaving two counters available to the firmware user. These timers may be used for a variety of on-board and off-board functions including timed-interval DMA requests and terminations or fail safe time out control for I/O operations.

SPECIFICATIONS

8089 IOP

WORD SIZE

Instruction — 16 to 40-bits

Data — 8, 16-bits

SYSTEM CLOCK

5.0 MHz \pm 0.1%

CYCLE TIME

2.2 microseconds for the fastest instructions

System Access Time

Dual-port Memory — 550 nanoseconds (worst case, without contention from on-board access)

I/O Capacity

MULTICHANNEL I/O Bus — 1 MULTICHANNEL port which supports 8 and 16-bit transfers and can be configured as a Basic Talker/Listener, Controller or Supervisor

System Development Capabilities

For applications where it is necessary to extend the User Command Firmware Package by writing additional 8089 IOP assembly language code, the development cycle can be significantly reduced and simplified by using the Intellec Series Microcomputer Development Systems. The 8089 IOP Software Support Package which includes a Macro assembler, linker, locator and PROM mapper is supported by the ISIS-II disk-based operating system.

In-Circuit Emulator

The ICE-86A or ICE-86 and ICE-86U upgrade kit provide the necessary link between the software development environment provided by the Intellec system and the "target" iSBC 589 execution system. In addition to providing a mechanism for loading executable code and data into the iSBC 589 board, the In-Circuit Emulator provides a sophisticated command set to assist in debugging software and in final integration of the user hardware and software.

iSBX™ MULTIMODULE™ — Two (2) iSBX MULTIMODULE boards

I/O Addressing

Interface	I/O Addresses
iSBX Connector #1	FF80 thru FF9F
iSBX Connector #2	FFA0 thru FFBF
MULTICHANNEL	FFD0 thru FFEE
Interval Timer	FFC8 thru FFCE
Other On-board Devices	FFC0 thru FFC6 FFF0 thru FFFE

Memory Capacity

ON-BOARD EPROM

Device	Total Capacity	Address Range
2716	8K bytes	FE000-FFFFFH
2732A	16K bytes	FC000-FFFFFH
2764	32K bytes	F8000-FFFFFH

ON-BOARD RAM

Total Capacity — 8K bytes

On-Board Address — 00000-01FFFFH

MULTIBUS® Address — Jumper selectable on 8K byte boundaries. Default is 0H.

I/O Transfer Rates (microseconds/transfer)

	MULTICHANNEL	iSBX™	MULTIBUS®		On-Board RAM
			Shared	Buslock	
MULTICHANNEL	—	2.0	2.4	2.2	1.8
iSBX	2.0	2.0	2.4	2.2	2.0
MULTIBUS (Shared)	2.4	2.4	2.8	—	2.2
MULTIBUS (Buslock)	2.2	2.2	—	2.4	2.0
On-Board RAM	1.8	1.8	2.2	2.0	1.6

Timers

Input Frequencies — Jumper selectable at 1.25 MHz, 625 KHz or 312.5 KHz

Output Frequencies/Timing Intervals —

Function	Single Timer/Counter		Dual Timer/Counter (Two Timers Cascaded)	
	Minimum	Maximum	Minimum	Maximum
Real-time delay	1.6 usec	210 msec	3.2 usec	1.37×10^4 sec
Programmable one-shot	1.6 usec	210 msec	3.2 usec	1.37×10^4 sec
Rate generator	4.76 Hz	625 KHz	7.3×10^{-5} Hz	312.5 KHz
Square-wave rate generator	4.76 Hz	625 KHz	7.3×10^{-5} Hz	312.5 KHz
Software triggered strobe	1.6 usec	210 msec	3.2 usec	1.37×10^4 sec
Hardware triggered strobe	1.6 usec	210 msec	3.2 usec	1.37×10^4 sec

Connectors

Interface	Double-Sided Pins (qty.)	Centers (in.)	Mating Connectors*
MULTIBUS System Bus	86	0.156	ELFAB BS1562043PBB Viking 2KH43/9AMK12 Soldered PCB Mount EDAC 337086540201 ELFAB BW1562D43PBB EDAC 337086540202 ELFAB BW1562A43PBB Wire Wrap
Auxiliary Bus	60	0.100	EDAC 345060524802 ELFAB BS1020A30PBB EDAC 345060540201 ELFAB BW1020D30PBB Wire Wrap
iSBX Bus (2)	36	0.100	iSBX 960-5
MULTICHANNEL Bus	60	0.100	3M 3334-6000 BERG 65949-960

* NOTE: Connectors compatible with those listed may also be used.

Interfaces

MULTIBUS® — All signals TTL compatible

MULTICHANNEL — All signals TTL compatible

iSBX™ Bus — All signals TTL compatible

Timers — All signals TTL compatible

Auxiliary Power/Memory Protect

There is no provision made on the iSBC 589 board for battery backup of RAM or for power fail detection.

MULTIBUS® Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-state	32
Address	Tri-state	32
Commands	Tri-state	32

Physical Characteristics

Width — 12.00 in (30.48 cm)

Height — 7.05 in (17.9 cm)

Depth — .50 in (1.27 cm)

Weight — 16 oz (453.6 gm)

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Relative Humidity — to 90% (without condensation)

Electrical Characteristics

DC POWER REQUIREMENTS

Configuration	Current Requirements (+ 5V + 5% maximum)
Without EPROM	4.7 amps
With 8K EPROM (using four 2716s)	5.4 amps
With 8K EPROM * (using two 2732As)	5.0 amps
With 16K EPROM (using four 2732As)	5.3 amps
With 32K EPROM (using four 2164s)	5.3 amps

* Factory default configuration

Reference Manuals

142996-001 — iSBC 589 Intelligent DMA Controller Board Hardware Reference Manual (Not Supplied)

142686-001 — Intel iSBX Bus Specification (Not Supplied)

143269-001 — Intel MULTICHANNEL Bus Specification (Not Supplied)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051

ORDERING INFORMATION

Part Number	Description
SBC 589	Intelligent DMA Controller Board

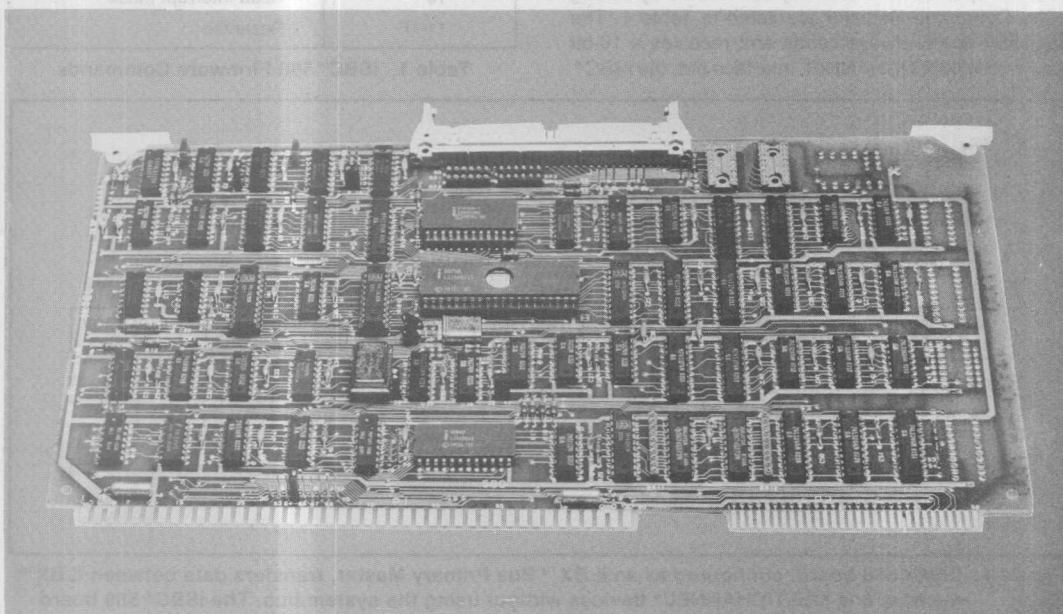
Part Number	Description	Center (in)	Double-sided Pins (pt)	Interface
EDAC 337884051	EDAC 337884051	0.125	88	MULTIBUS System Bus
EDAC 337884052	EDAC 337884052	0.100	60	Auxiliary Bus
EDAC 3458054801	EDAC 3458054801	0.100	38	iSBX Bus (2)
EDAC 3458054802	EDAC 3458054802	0.100	60	MULTICHANNEL Bus

NOTE: Connector nomenclature with those listed may also be used.

iSBC® 580 MULTICHANNEL™ BUS TO iLBX™ BUS INTERFACE

- **MULTICHANNEL™ I/O bus 16-bit
Talker/Listener interface**
- **iLBX™ bus master interface (primary or
secondary)**
- **Supports MULTIBUS® interrupts**
- **Data rates up to 5.3 megabytes per
second**
- **Addresses up to 16 megabytes of
iLBX™ bus memory**
- **MULTIBUS® form factor**

The iSBC® 580 Interface Board is a member of Intel's complete line of MULTIBUS® microcomputers which maximize system performance by using separate optimized buses for intra-system communication (MULTIBUS system bus), high speed I/O (MULTICHANNEL™ DMA I/O bus), expansion I/O (iSBX™ I/O expansion bus) and high-speed memory expansion (iLBX™ execution bus). The iSBC 580 board provides a key element in the enhanced MULTIBUS system architecture by implementing a MULTICHANNEL I/O bus to iLBX bus interface on a single 6.75 x 12.00 inch printed circuit board. Using an LSI state machine with standard on-chip firmware to maximize throughput, the on-board Intel® 8048 Single Component Microcomputer transfers data between a MULTICHANNEL Controller, device and up to 16 megabytes of iLBX bus resident memory at rates up to 5.3 megabytes per second. Acting as a MULTICHANNEL Talker/Listener, the iSBC 580 board increases the system's overall performance by transferring data between the MULTICHANNEL I/O bus and system memory without using the MULTIBUS system bus. As shown in Figure 1, this allows other system tasks to utilize MULTIBUS resources while high-speed I/O block transfers are occurring simultaneously. The board's high throughput and independence from MULTIBUS activities make it an ideal solution for applications that must transfer large amounts of data in and out of a MULTIBUS system, such as MULTIBUS to host computer links and mass storage, graphics display and high-speed data acquisition subsystem interfaces.



FUNCTIONAL DESCRIPTION

MULTICHANNEL™ Interface Capabilities

The MULTICHANNEL I/O bus is designed to provide a general purpose, high-speed data path between a microcomputer system and up to 15 block transfer devices. Using a 16-bit wide data bus and a simple asynchronous handshaking scheme, the MULTICHANNEL bus can operate over distances up to 15 meters (50 feet) with a maximum burst throughput of 8 megabytes/second. The bus consists of 16 address/data lines, 6 control lines, 2 interrupt lines, parity lines and reset. Via these signals, a MULTICHANNEL Supervisor or Controller may configure and then initiate a block data transfer with any other device on the bus.

The iSBC 580 board acts as a 16-bit only Talker/Listener device on the MULTICHANNEL I/O bus. As a Talker/Listener, the board will respond to Register Read or Write and DMA requests issued by the MULTICHANNEL Supervisor (typically an iSBC 589 board) or by a MULTICHANNEL Controller device.

The iSBC 580 board implements 32 MULTICHANNEL Device Registers. The first three registers are the standard STO Status, SRQ Status and SRQ Mask Registers, as defined by the MULTICHANNEL Bus Specification. The remaining registers are used to communicate with the on-board firmware and for user data storage. The firmware operations which may be initiated by writing to the Command Register are listed in Table 1. The iSBC 580 board always sends and receives a 16-bit word on the MULTICHANNEL interface but, the iSBC®

580 device registers (see Table 2) are 8-bit only. Register Write operations use only the low order 8-bits (AD0-AD7). Register Read operations place the data on the low order data lines of the MULTICHANNEL I/O bus and set the high order data lines to FFH.

Command Code (Hex)	Operation
0	No Operation
1	Go off line forever
2	STO poll (diagnostic)
3	SRQ poll (diagnostic)
4	Set on-board timer
5	Read on-board timer
6	Start on-board timer
7	Stop on-board timer
8	Generate Task Complete interrupt
9	Perform checksum on firmware (diagnostic)
A	Turn on-board LED on
B	Turn on-board LED off
C	Reset
D, E	Reserved
F	Set interrupt mask
10	Read interrupt mask
11-1F	Reserved

Table 1. iSBC® 580 Firmware Commands

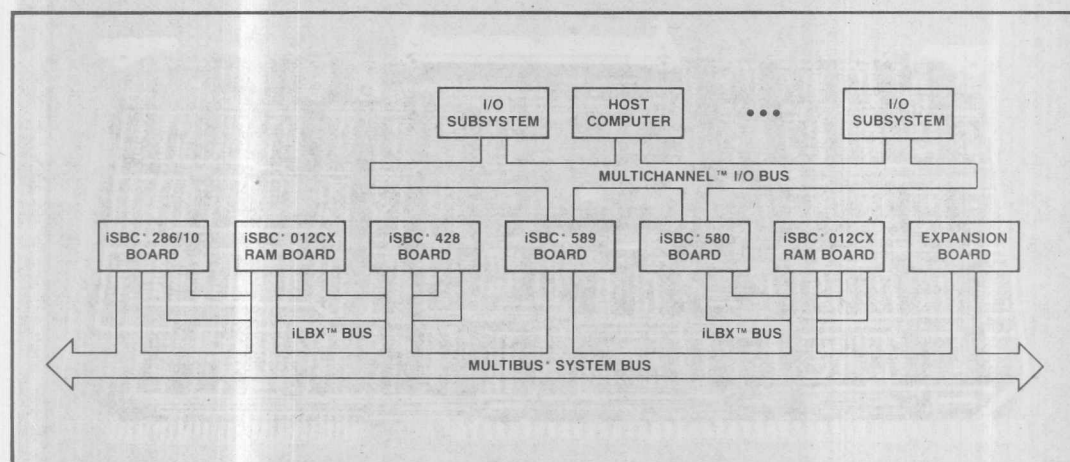


Figure 1. iSBC® 580 board, configured as an iLBX™ Bus Primary Master, transfers data between iLBX™ memory and MULTICHANNEL™ devices without using the system bus. The iSBC® 589 board acts as the MULTICHANNEL™ Supervisor and performs data transfers between MULTIBUS® memory and MULTICHANNEL™ devices.

The iSBC 580 board can generate maskable MULTICHANNEL STO interrupts when the board detects a parity error in incoming MULTICHANNEL data, when the board attempts to address non-existent iLBX memory or when the board detects a MULTIBUS interrupt from the system in which it resides. The last type of interrupt allows a single board computer to send an interrupt via the iSBC 580 board to the MULTICHANNEL Supervisor located in another MULTIBUS system. The board can also generate a number of SRQ interrupts on the MULTICHANNEL bus as shown in Figure 2.

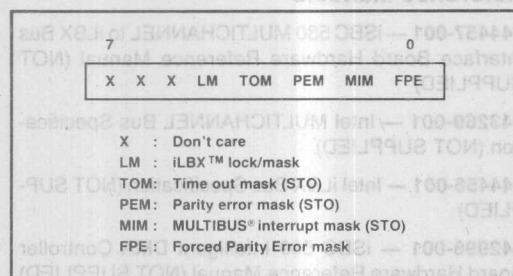


Figure 2. iSBC® 580 Interrupt Mask Register (14H)

iLBX™ Bus Interface Capabilities

Used in conjunction with the MULTIBUS interface, the iLBX bus is designed to provide off-board memory and I/O expansion for single board computers while maintaining on-board performance. The iLBX bus provides high-speed access to compatible expansion boards by granting privileged use of the bus to a single Primary Master. The bus also provides limited access to iLBX bus expansion boards for, at most, one Secondary Mas-

ter that requires only occasional or non-concurrent access to iLBX resources. The iLBX bus, with 16 data lines, 24 address lines plus control, parity and interrupt signals, utilizes all the pins on the P2 connector except the four pins dedicated to the high-order address lines of the MULTIBUS interface. The non-multiplexed address and data lines provide access to up to 16 megabytes of iLBX bus resident memory, on up to 4 separate expansion boards, at speeds comparable to that of a single board computer's on-board resources.

The iSBC 580 board is configurable as either a Primary or a Secondary Master on the iLBX bus. Figure 1 shows a typical system configuration, with an iSBC 580 board acting as a Primary Master. The board can access up to 16 megabytes of iLBX memory. Supporting 16-bit transfers on the MULTICHANNEL bus, the board accesses memory as 16-bit words on even byte iLBX address boundaries. To increase the performance of iLBX memory read operations, the iSBC 580 board prefetches data from memory while the current data word is being transferred over the MULTICHANNEL I/O bus.

Register	Address
STO Status	00H
SRQ Status	01H
SRQ Mask	02H
RESERVED	03H-0FH
General Purpose Registers	10H*-1FH

* NOTE: 10H used as Command Register.

Table 2. iSBC® 580 MULTICHANNEL™ Device Register Set

SPECIFICATIONS

MULTICHANNEL™ Bus

Interface — Basic Talker/Listener

Transfer Mode — 16-bit

Device Address — Jumper selectable between 00H and 0EH

Registers — STO status, SRQ status, SRQ mask plus device specific registers

Signal Level — TTL compatible

iLBX™ Bus

Interface — Primary or Secondary (default) Master

Transfer Mode — 16-bit

Addressing — 16 megabytes on even byte boundaries only

Signal Level — TTL compatible

MULTIBUS® Interface

Data — None

Addressing — None

Interrupts — Jumper configurable to use any 1 of the 8 MULTIBUS interrupt lines. Interrupts are edge triggered.

Signal Level — TTL compatible

Throughput

5.3 megabytes/sec (2.65 megatransfers) max.

Connectors

iLBX™ BUS INTERFACE

Double-Sided Pins — 60

Centers — 0.100 in.

Mating Connectors* — Kelam RF30-2803-5
T&B Ansley A3020
(609-6025 modified)

MULTICHANNEL™ BUS INTERFACE

Pins — 60

Centers — 0.100 in.

Mating Connectors* — 3M 3334-6000
Berg 65949-960

* Connectors compatible with those listed may also be used.

Physical Characteristics

Width — 12.00 inches (30.5 cm)

Height — 6.75 inches (17.1 cm)

Depth — 0.60 inches (1.5 cm)

Weight — 12 ounces (340 gm)

ORDERING INFORMATION

Part Number Description

SBC 580	MULTICHANNEL to iLBX Bus Interface Board
---------	--

Environmental Characteristics

Operating Temperature — 0° to 55°C

Relative Humidity — to 90% (without condensation)

DC Power Requirements

Voltage — +5 volt only $\pm 5\%$

Current — 2.5 amps (typical)

Reference Manuals

144457-001 — iSBC 580 MULTICHANNEL to iLBX Bus Interface Board Hardware Reference Manual (NOT SUPPLIED)

143269-001 — Intel MULTICHANNEL Bus Specification (NOT SUPPLIED)

144456-001 — Intel iLBX Bus Specification (NOT SUPPLIED)

142996-001 — iSBC 589 Intelligent DMA Controller Board Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051

iSBC® 570, 576, 577 INTEL SPEECH TRANSACTION FAMILY

- **Friendly man-machine interface—**speech is the most natural and most easily learned form of interaction for man.
- **Lower data entry cost—source data capture**
- **Higher accuracy—operator mental encoding is eliminated.**
- **Freedom of Movement—More efficient work flow**
- **Hands and eyes free—ability to perform another primary task**
- **Easier training—interactive, generic terminology**
- **Complements keyboard/CRT—new dimension to data entry**

Users world wide are recognizing the many advantages of having Automatic Speech Recognition (ASR) and Electronic Speech Synthesis (ESS) in their products and applications. Speech I/O is a new dimension in data entry/control that complements other I/O mechanisms.

Speech I/O as a direct man-machine interface can be used for a broad range of applications, such as office and factory automation, computer-aided design, QC inspection stations, inventory control—and many more. Whatever your application is, the benefits of speech I/O are measured in dollars saved, improved productivity and improved product quality.



The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products: BXP, CREDIT, i, ICE, ICS, Im, Insite, Intel, INTEL, Intelevison, Intellec, IMMX, IOSP, IPDS, IRMX, ISBC, ISBX, Library Manager, MCS, MULTIMODULE, Megachassis, Micromainframe, Micromap, MULTIBUS, Multichannel, Plug-A-Bubble, PROMPT, Promware, RMX/80, System 2000, UPI, and the combination of ICS, IRMX, ISBC, ISBX, ICE, MCS, or UPI and a numerical suffix. Intel Corporation Assumes No Responsibility for the use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Patent Licenses are implied. ©INTEL CORPORATION, 1982

In computer-aided design and manufacturing (CAD/CAM), design commands by speech allow the design engineer to keep his attention focused on the actual graphic elements.

In manufacturing, speech transactions provide important advantages in productivity. Defect tracing, production line monitoring and synchronization, and factory data collection, all benefit from direct human speech to computer communication.

In the automated office, ever-increasing machine intelligence can be controlled without mastering of typing skills.

The basic concept of a speech I/O system is shown in Figure 1. The speech I/O system provides a human-oriented interface with a machine-oriented computer-based information system or process. The speech I/O system recognizes speech inputs, provides visual/audio prompts and verification, and handles message editing and buffering. Depending on what was recognized, digitally coded data is then used to interact with the machine-oriented computer-based system.

The functional blocks of a speech I/O system are shown in Figure 2.

A complete system includes not just the capabilities for signal conditioning, Automatic Speech Recognition (ASR), and Electronic Speech Synthesis (ESS), but must include speech transaction processing as well. The Speech Transaction Processing task includes:

- The conversion between spoken language and coded representation
- Operator prompting and feedback
- Message editing
- Message buffering

In addition, development tools should be available for the generation of speech transaction files that will define the operations of the speech I/O system. Figure 3 shows the function of each member of the Intel Speech Transaction Family.

The Intel Speech Transaction Family, iSBC® 570, iSBC® 576 and iSBC® 577, is a family of products that provides a minimal risk path to add speech Input/Output (I/O) to your product line. The Speech Transaction Family will allow you to move from evaluation to integral speech driven products without major redesigns. Depending on your stage of product development, whether it is an evaluation, or a product simulation, or an add-on speech option, or a

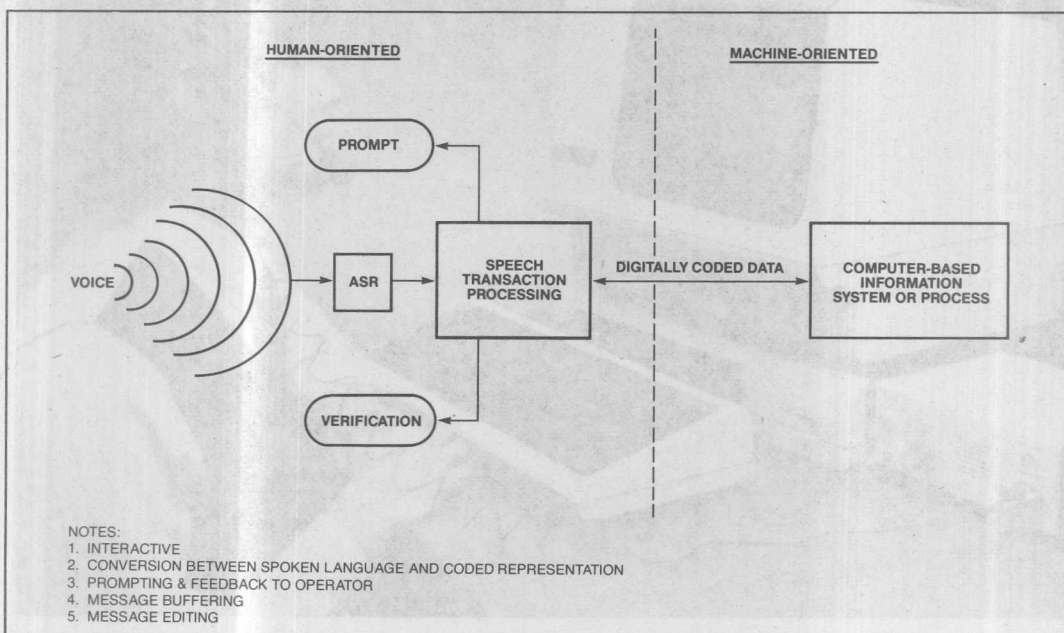


Figure 1. Basic Concept

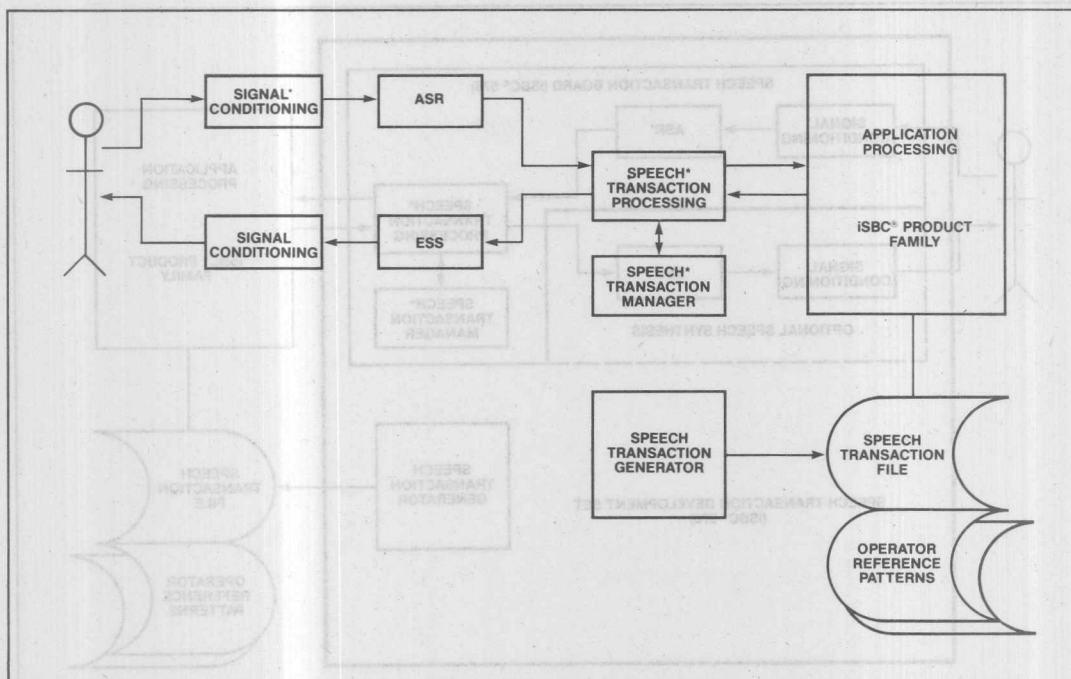


Figure 2. Functional Blocks of Speech I/O System.

fully integrated speech product, the Speech Transaction Family's flexibility allows your speech I/O application to grow with a minimal amount of engineering effort. The Speech Transaction Family allows you to adapt your product to various markets as your application needs change, without a major redesign. Whether it is a configured speech development system, or easy-to-integrate speech board, or a maximum value-added speech component chip set, an Intel product is ready to meet your needs.

Development of your speech I/O system may have been your stumbling block in the past. The requirement for speech technology expertise, extensive hardware development and extensive software development are a thing of the past. Integral to the Speech Transaction Family are highly sophisticated computer-based design and development tools that will take you from product concept to a working speech product with a minimal effort. In-depth knowledge of speech algorithms and of speech human factors considerations are no longer an absolute requirement of your system designers.

Intel provides the total solution. Speech hardware has been designed to work with our wide selection of Multibus® single-board computers, memory cards, and data I/O cards. Speech software is based on the Real-Time Multi-Tasking Executive (RMX-88). Speech transaction software development has been implemented on our universal Intellect® Microcomputer Development System. All of the pieces have been engineered to provide an easily integrated speech I/O solution.

Speech I/O is a new technology area. Intel has developed a family of products and services, that will fit your development sequence needs for a new technology with minimal risk and ease of use. A very likely evaluation and development sequence you may follow is illustrated in Figure 4 and Figure 5 along with Intel's products and services that are offered to meet those needs. Having products and services that can satisfy the illustrated sequence is very important in reducing the risk, engineering cost, and lowering incremental investments necessary as product requirements change.

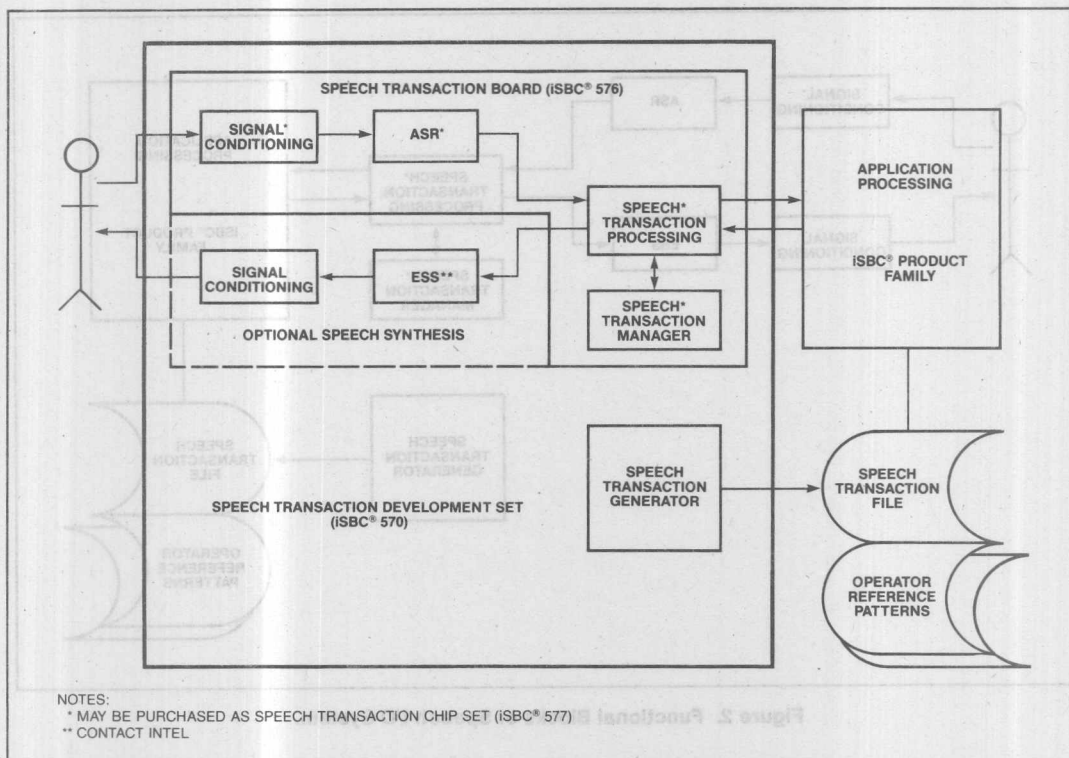


Figure 3. Functional Blocks of the Intel Speech Transaction Family.

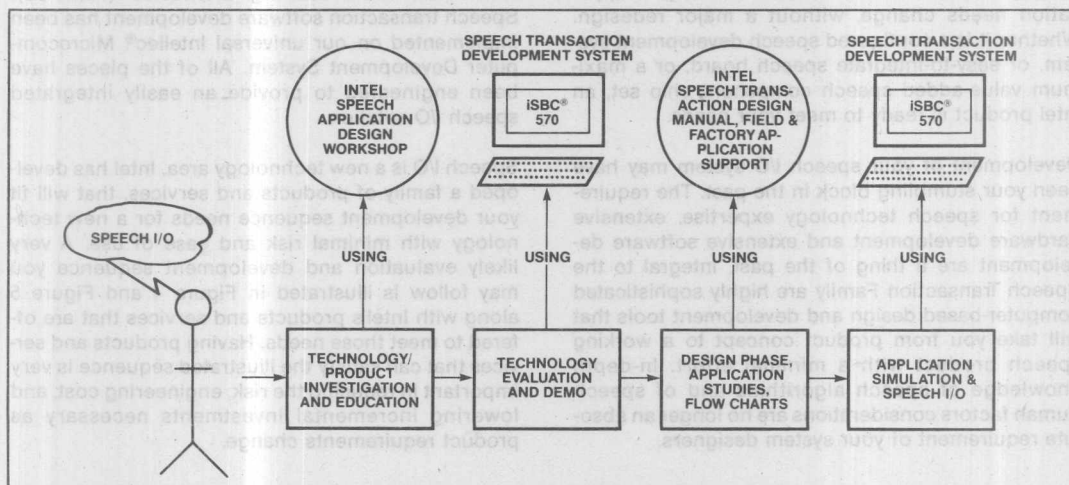


Figure 4. Application Definition Phases

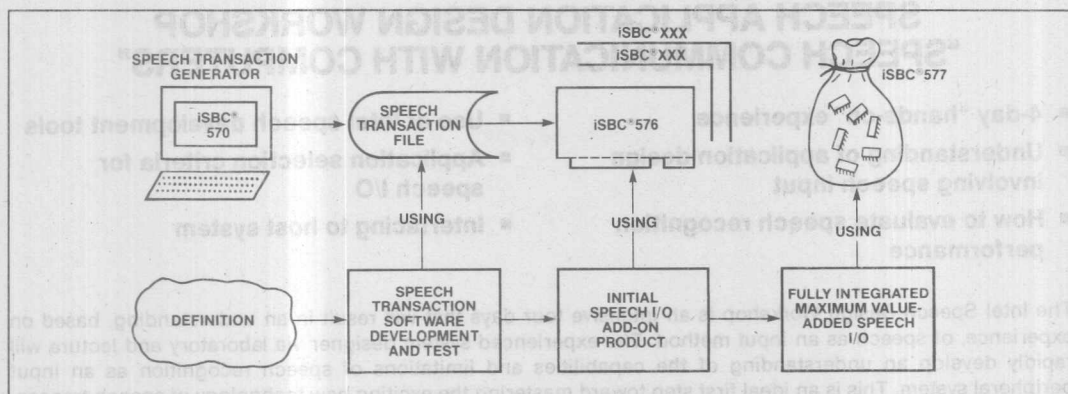


Figure 5. Application Implementation Phases

The sequence starts with a workshop to learn about the Speech Technology and to develop a necessary knowledge base to evaluate potential applications. The next stage, an evaluation-oriented Speech Transaction Development System (iSBC® 570 and Inteltec® Microcomputer Development System), provides technology evaluation and demonstrations without engineering investment. Using the experience from the two previous stages, plus field and factory application support, the design phase can now proceed. Once the application framework has been established, application simulation can be performed using the Speech Transaction Development System.

Upon successful completion of simulation, the speech transaction software development can be easily completed on the same Speech Transaction Development System. The initial speech I/O products can then be shipped using the Speech Transaction Board (iSBC® 576). When higher volume justifies increasing the value added, the chip set, iSBC® 577, can be used. Throughout the process, whether it is system, board, or chip set, the same software is utilized. Very little is lost as your product needs change. The level of investment required tracks the stage of product development. Your risk and exposure is kept to a minimum.

SPEECH APPLICATION DESIGN WORKSHOP "SPEECH COMMUNICATION WITH COMPUTERS"

- 4-day "hands-on" experience
- Understanding of application design involving speech input
- How to evaluate speech recognition performance
- Use of Intel speech development tools
- Application selection criteria for speech I/O
- Interfacing to host system

The Intel Speech Design Workshop is an intensive four days that will result in an understanding, based on experience, of speech as an input method. The experienced system designer via laboratory and lecture will rapidly develop an understanding of the capabilities and limitations of speech recognition as an input peripheral system. This is an ideal first step toward mastering the exciting new technology of speech transaction processing. All persons directly involved in the selection or implementation of a speech recognition-based system should attend. Attendees are presumed to be familiar with computers, a programming language, and the Intellec® Microcomputer Development System. The student will experience all phases of solution development by designing, implementing, testing, and presenting to the class an extension to the basic speech demonstration provided with the Speech Transaction Development Set.

DAY 1

What's unique about speech?—Communicating with computers using words and phrases rather than keystrokes.

Overview of the Intel Speech Transaction Family.

Introduction to the Speech Transaction Board (STB), Speech Transaction Manager (STM), Speech Transaction File (STF), and the Speech Transaction Generator (STG).

Lab: Install iSBC® 570 Speech Transaction Development Set. Operate Speech Training Demo Program.

The 7 modes of operation of the STM.

Lab: Using the Evaluation mode to ascertain performance.

DAY 2

The Speech Transaction Generator (STG).

The Speech Transaction File (STF) using the training demonstration program as an example.

Application selection criteria.

Transaction design.

Lab: Develop an extension of the training demonstration program using the STG.

DAY 3

Human factors considerations—case histories.

Operator training and performance evaluation.

Automatic Speech Recognition (ASR) subsystem parameters.

Lab: Evaluate performance of new STF developed on Day 2. Measure effects of ASR parameter changes on recognition.

DAY 4

Configuring the application solution.

Host I/O interfacing.

Host program design concepts.

Lab: Discuss student-developed program extensions.

iSBC® 570 SPEECH TRANSACTION DEVELOPMENT SET

■ Complete Development Support Set for the Intel Speech Product Family.

Includes:

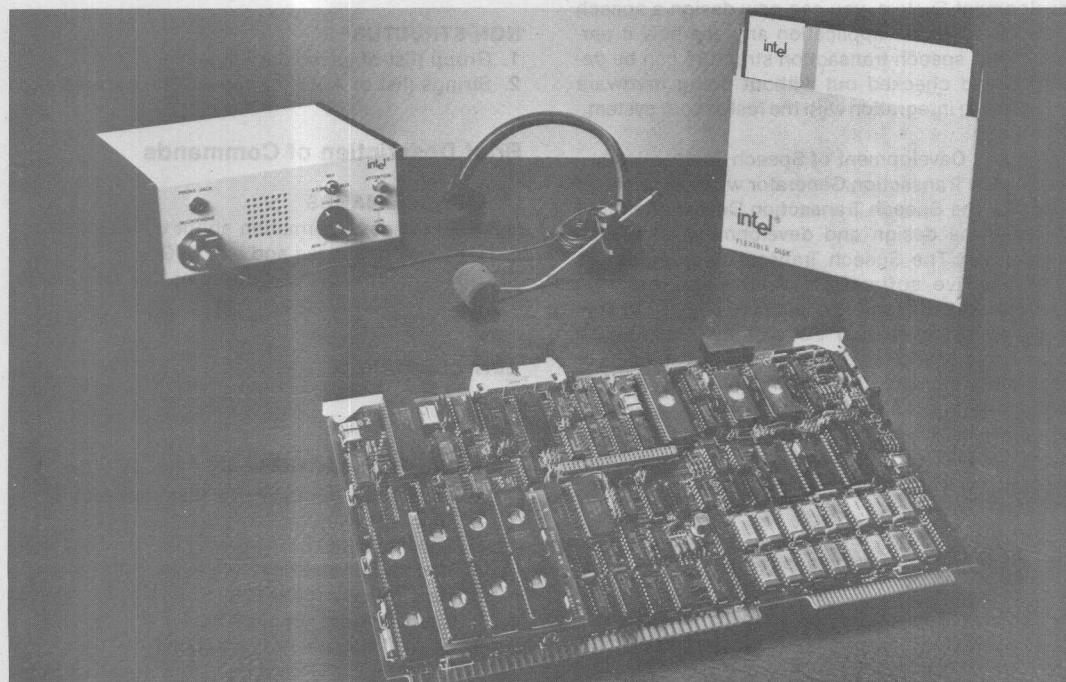
- Speech Transaction Generator
- iSBC® 576 Speech Transaction Board
- iSBC® 575 Operator Control Unit
- Microphone
- Demo program
- Speech Transaction Design Manual

■ Intel® Microcomputer Development System based

The Speech Transaction Development Set, iSBC® 570, provides an easy-to-use package for speech transaction evaluation, design simulation and application development. Along with Intel's Speech Design Workshop, the Speech Transaction Development Set becomes the starter kit that will move you into the forefront of speech I/O systems. Using the demo program supplied, you are quickly introduced to the important attributes of speech. Using the Intel® Microcomputer Development System and writing/modifying software based on examples provided, you can quickly simulate your application without hardware development. And finally, with the Speech Transaction Generator, your speech transaction structure, definition, transaction file coding and management become a well-defined automated task.

■ Speech Transaction Generator provides:

- Interactive design environment
- A speech transaction structure embodying good human factors engineering
- Automatic error checking of transaction design
- Symbolic labeling for easy system designer reference
- Speech Transaction File data base manager facilitates Speech Transaction File changes



FUNCTIONAL DESCRIPTION

The iSBC® 570 Speech Transaction Development Set has been designed to meet your speech I/O needs as your level of involvement with speech I/O system grows. The Speech Transaction Development Set serves three very important functions. The three functions are: 1) Technology Evaluation and Demonstration, 2) Application Simulation of Speech I/O, and 3) Design and Development of Speech Transaction Software. These three functions are discussed below.

Technology Evaluation and Demonstration — A complete demo package is provided for you to demonstrate the capabilities of speech I/O. This package allows you to evaluate the speech technology without investing engineering design and development time. It is easy to use. Major attributes of a speech I/O system are highlighted and fully documented. The host system for the demonstration is the Intellec® Microcomputer Development System.

Application Simulation of Speech I/O—The Speech Transaction Development Set provides the necessary tools and program examples for you to easily simulate your speech I/O system using the Intellec® Microcomputer Development System as the host. With the iSBC® 570 and the Intellec® Microcomputer Development System, you can now design a speech I/O system for your application and see how it performs. Your speech transaction structure can be developed and checked out without doing hardware and software integration with the rest of your system.

Design and Development of Speech Transactions — The Speech Transaction Generator which is provided as part of the Speech Transaction Development Set facilitates the design and development of speech transactions. The Speech Transaction Generator is an interactive software development tool that generates the Speech Transaction File (STF) that configures your speech I/O system. The Speech Transaction Generator checks for inconsistencies or incomplete transactions. The generated code is guaranteed to be fully compatible with the Speech Transaction Board. The Speech Transaction Generator will not only shorten your development time, but will also facilitate a well human-engineered speech I/O interface.

OPERATIONAL DESCRIPTION

The Speech Transaction Generator is implemented in two parts. The first part is the processing element

of the STG and resides on EPROM in an STB environment. The second part is the data base manager for the STG and resides as an executable file under ISIS. The STG allows a system designer (with appropriate knowledge of transaction, fields, vocabulary and synthesis) to specify a STF easily. The STG maintains a set of files on the Intellec® system as the data base. In this manner, the STG is the customization tool used by the speech system designers to prepare application-unique speech transactions that will execute on the STB under the supervision of the Speech Transaction Manager (STM). The STG also allows the system designer to dump portions of this data base in an ASCII-text format to a file. This ASCII-text file is useful for transporting data base entries between the STG implemented on other than an ISIS environment.

The things that a system designer can manipulate with the STG are termed "objects." Objects can be categorized into structures and non-structures. Structures are generally a string of characters or a list of tags. Objects are classified as follows:

STRUCTURES

1. Transaction
2. Fields
3. Vocabulary
4. Synthesis

NON-STRUCTURES

1. Group (list of vocabulary tags)
2. Strings (list of ASCII or non-ASCII characters)

Brief Description of Commands

UTILITY COMMANDS

HELP—Provides information about the objects

EXIT—Close data base and exit STG

PREFIX—Specify prefix character for DEFine or MODify commands

EDIT COMMANDS

DEFine

DEFINE TRANSACTION:

1. Vocabulary tag to enable this transaction?
2. Training group?
3. Starting field?
4. Host buffer strategy?
5. Verification actions?
6. Special reject actions?
7. Special illegal function action?

DEFINE FIELD:

1. Prompt?
2. Help message?
3. Prefix for host message?
4. Suffix for host message?
5. Special functions enable?
6. Valid sources?
7. Multiple utterance path?
 - If yes,
 - a) Vocabulary words?
 - b) Next field?
 - c) Maximum number of utterances?
 - d) Fixed or variable?
8. Vocabulary words?
9. Next field?

DEFINE VOCABULARY:

1. Name?
2. Visual verify?
3. Audio verify?
4. Host message?
5. Visual train?
6. Audio train?
7. Special functions?

DEFINE SYNTHESIS:

1. Function?
2. Duration?
3. Delay?

DELeTe

Removes objects from the data base.

MODIfy

Modifies objects already entered into the data base with the DEFine command.

SPECIFICATIONS

Operating Environment

Intellec® Microcomputer Development System (Model 800, Series II, and Series III with 64K byte of RAM).

SUPPLIED EQUIPMENT

iSBC® 576—Speech Transaction Board with Speech Transaction Manager Firmware.

VALIDATION AND GENERATION COMMANDS

VALIdate

Sequences through each of the transactions specified and validates them for completeness and proper definition.

GENerate

Takes the result of a successful validate command and produces a memory image of the STF. The STF can now be executed.

INTERROGATION COMMANDS

DISPlays

Displays the contents of the objects

LISTs

Lists the directory of the objects

FILE INTERFACE COMMANDS

DUMp

Passes results of current validation and outputs it to the host in a .DMP file.

USE

Takes command input from the specified file.

- Speech Transaction Generator software and firmware.
- Speech I/O Demo Software.
- iSBC® 575 Operator Control Unit.
- Shure SM-10A Microphone.
- Speech Transaction Design Manual.

OPTIONAL EQUIPMENT

- iSBX®-351—RS232 Multimodule
- iSBC®-342—EPROM expansion module
- SBX synthesizers

ORDERING INFORMATION

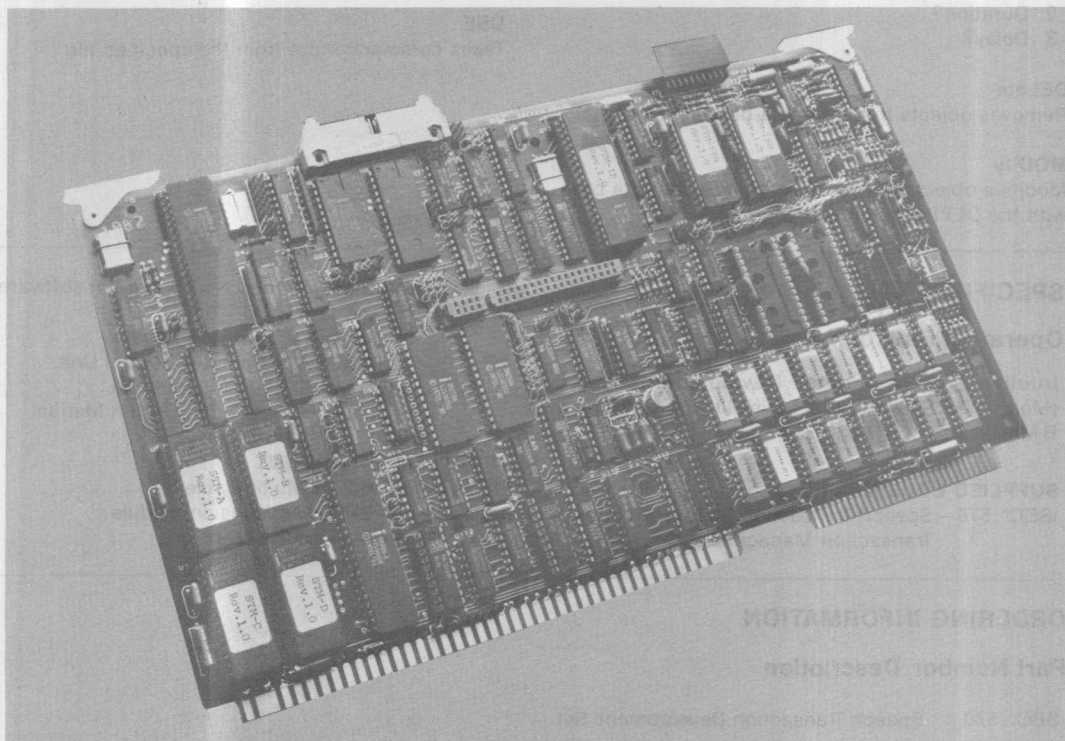
Part Number Description

iSBC® 570 Speech Transaction Development Set

iSBC® 576 SPEECH TRANSACTION BOARD

- Up to 200 recognition words or phrases
- Automatic ASR and ESS handling
- On-board Speech Transaction Manager
- 8086, 16-bit CPU
- On-board diagnostic
- Multibus or serial host interface
- iSBX® interface
- Built-in buffer editing functions

The iSBC® 576 Speech Transaction Board is the heart of a speech I/O system. Beside providing Automatic Speech Recognition (ASR) capabilities, a ROM-resident Speech Transaction Manager (STM) is included on the board. This provides a flexible operating structure for the system designer with a fully buffered speech-generated input-transaction handling capability. Flexibility has been designed into the STM to allow integration into existing applications without a major rewrite/redesign of host application software and hardware. The Speech Transaction Manager accommodates a Speech Transaction File which configures the iSBC® 576 Speech Transaction Board for each application. Also included on the board are three selectable audio feedback tones, visual feedback/control via a CRT terminal or printer, and an optional Electronic Speech Synthesis (ESS) capability.



FUNCTIONAL DESCRIPTION

Figure 6 shows the functional structure of the Speech Transaction Board.

Input Signal Conditioning—Microphone input signal is amplified and low-pass filtered. The conditioned signal is then digitized and passed through 16 band-pass digital filters implemented by 2920/21 analog signal processors. The 2920/21s are synchronized and are operating in parallel. The bandpass filter information is then assembled by an 8048 microcomputer for algorithm processing by an 8086 processor. System-to-system portability is guaranteed by the usage of digital signal processing techniques.

ASR—Automatic Speech Recognition is accomplished by the 8086 processor in conjunction with two 2920/21 digital signal processors and an 8048 microcomputer. ASR handling is done completely under the control of the Speech Transaction Manager. This task is transparent to the system designers. Automatic statistics are also provided to track system performance.

Tone Generator—3 audio tones are available for use as a prompt. The tones are generated within a 2920 analog signal processor. The tone generator also generates test patterns for use by the diagnostic section.

Diagnostic—Under the control of the Speech Transaction Manager, a diagnostic check of the speech

recognition hardware and software can be performed. System integrity is automatically determined to insure repeatable performance.

Output Signal Conditioning—Output amplifiers are provided to drive a speaker for the audio tones. Volume can be varied by a potentiometer.

Terminal Driver—Under the control of the Speech Transaction Manager, a CRT terminal/keyboard can be connected directly to the Speech Transaction Board. The terminal can be used for visual feedback as well as data entry/control. The interface is RS232C compatible.

Operator Control—Two LED lights to indicate recognition status and an operator attention button are provided. These functions are programmable under the control of the Speech Transaction Manager.

Operator Reference Patterns—Speech patterns for recognition are normally contained in RAM. The patterns are downloaded from the host processor under the control of the Speech Transaction Manager. The operator reference patterns are also generated under the control of the Speech Transaction Manager.

Speech Transaction Manager—The Speech Transaction Manager is the heart of the Speech Transaction Board. The Speech Transaction Manager controls all of the functions within the board. This firmware is

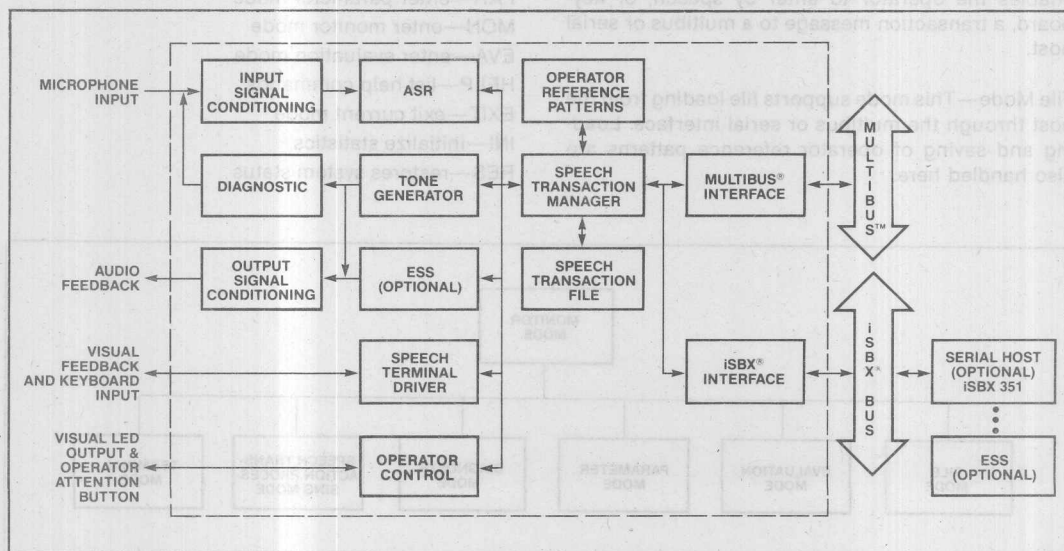


Figure 6. Functional Structure of the Speech Transaction Board

contained in 27128 EPROMs and is RMX®-88 (Real-Time Multi-Tasking Executive) based. Processing is provided by the 8086 processor.

Speech Transaction File—The Speech Transaction File determines the configuration of the board for each application. The Speech Transaction Manager executes this file which is normally downloaded from the host and stored in RAM. The file can also be stored in ROM/EPROM on the Speech Transaction Board itself. These files are generated by the Speech Transaction Generator.

Multibus® Interface—A slave multibus® interface is implemented. On the multibus the Speech Transaction Board looks like a data port.

ISBX® Interface—One SBX® interface has been implemented. This interface is controlled by the Speech Transaction Manager. Interface with a non-Multibus® host can be implemented via this channel.

OPERATIONAL DESCRIPTION

The operation of the Speech Transaction Board is determined by the Speech Transaction Manager. The Speech Transaction Manager has several specific modes of operation as described below.

Speech Transaction Processing Mode—This mode enables the operator to enter by speech, or keyboard, a transaction message to a multibus or serial host.

File Mode—This mode supports file loading from the host through the multibus or serial interface. Loading and saving of operator reference patterns are also handled here.

Diagnostic Mode—This mode tests the hardware. The diagnostics will test the 2920/8048 interface and the 8048/8086 interface.

Terminal Mode—This mode provides for direct communication between the host and the Speech Transaction Board terminal. All response from the operator (through the terminal) is passed directly to the host. ALL host messages are passed directly to the terminal.

Parameter Mode—This mode lets the user define a limited set of configuration information and to set various other system parameters.

Evaluation Mode—This mode lets the user evaluate the recognition performance of an STF vocabulary or a vocabulary entered from the STB terminal. Use of this mode will facilitate evaluation of training strategies, vocabulary choices and parameter settings. In this mode statistics and automatic scoring of results are all standard features.

LIST OF COMMANDS

Monitor Mode Commands

STP—enter speech transaction processing mode

FIL—enter file mode

DIA—enter diagnostic mode

TER—enter terminal mode

PAR—enter parameter mode

MON—enter monitor mode

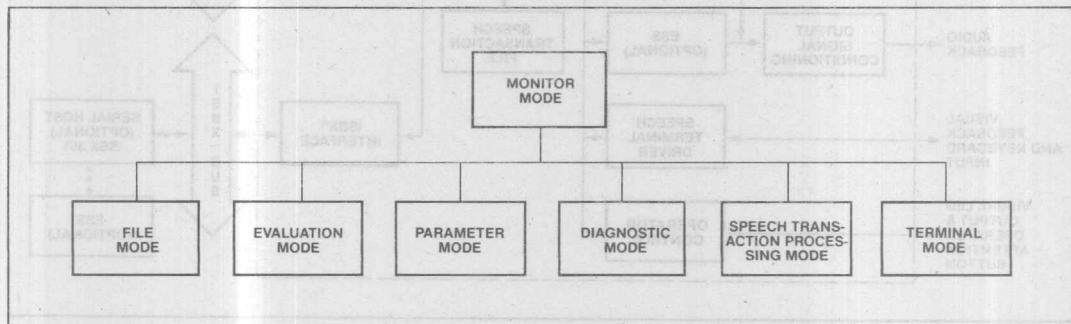
EVA—enter evaluation mode

HELP—list help commands

EXIT—exit current mode

INI—initialize statistics

RES—restores system status



Speech Transaction Processing Mode Function

Buffer Editing Functions

Forward	Erase Field
Backup	Continue
Correction	Beginning
Replace	Cancel
Forward Field	Finish
Backup Field	

Utility Functions

Help—operator assistance at each field
Display—current transaction buffer
Next—go to next field
Detach—put terminal in "Terminal Mode"
Attach—get terminal out of "Terminal Mode"
Exit—exit STP mode
Up—raise rejection threshold
Down—lower rejection threshold
Relax—put system in not-ready state
Ready—first of two utterances to exit not-ready state
Attention—second of two utterances to exit not-ready state
Enable Transaction "N"—initiate transaction
Macro—performs a series of commands automatically in any mode

Operator Speech Pattern Maintenance Functions

Test Group	Train
Test All	Train Group
Retrain	Train All
Retrain Group	Update
Retrain All	Update Group
Delete	Update All
Delete All	Test

File Mode Commands

LST—load Speech Transaction File
SST—save speech transaction file
LRP—load operator speech patterns
SRP—save operator speech patterns
CRP—clear operator speech pattern RAM area
HELp—list help commands
CST—clear speech transaction

EXIt—exit current mode
LDI—load dictionary
SDI—save dictionary

Diagnostic Commands

FET—front end test
EXIt—exit mode
HELp—list help commands

Parameter Mode Commands

BLO—block size of transfer
CHS—communication header
CON—display all configuration parameters
DIS—discrimination level
DRE—small delta rejection
EST—display extended statistics
HOS—specifies host and characteristics
HTE—host terminator string
HTO—host time-out
INS—initialize statistics
MTP—minimum training passes
RPT—operator reference pattern names
SHC—serial host baud rate
STA—displays statistics
STF—STF name
STR—ROM STF name
TST—STB terminal status
WRD—word gap and word length
FEG—front-end gain
HELp—list help commands
EXIt—exit current mode

Evaluation Mode Commands

DEF—define
MVO—modify vocabulary
RVO—remove vocabulary
RRP—remove reference pattern
RET—retrain
LIS—list vocabulary
TRAI—train
UPDate—update
TEST—test
RECOgnition—recognition
STA—statistics
COR—cross correlation
INS—initialize statistics
HELp—list help commands
EXIt—exit current mode

SPECIFICATIONS

Operating Environment

Host Processor—any iSBC® Multibus® computer
 —any RS232 serial host interface
 Audio Input—475Ω input impedance
 —50 m.v. p-p max.
 —differential or single-ended

Equipment Supplied

iSBC® 576 Speech Transaction Board with Speech
 Transaction Manager Firmware

Optional Equipment

iSBX®-351	RS232 Multimodule
iSBX®-342	EPROM expansion
	SBX synthesizer
iSBC®-575	Operator Control Unit

Performance Specifications

Recognition vocabulary—200 words or phrases
 Utterance duration—user selectable > 100 msec.,
 minimum
 —user selectable < 2 sec.,
 maximum

Rejection Threshold—user selectable

Word gap—user selectable > 50 msec., minimum
 —user selectable < 250 msec.,
 maximum

Recognition Accuracy (50 state names)—99+%

Response Time (for vocabulary up to 200 words
 with maximum node length 50
 words) — < 500 msec.

ORDERING INFORMATION

Part Number Description

iSBC® 576	Speech Transaction Board
-----------	--------------------------

Physical Characteristics

Width—6.75 in. (17.15 cm)
 Height—0.5 in. (1.27 cm)
 Length—12.0 in. (30.48 cm)
 Shipping weight—TBD
 Mounting—occupies one slot of iSBC® system
 chassis in cardcage/backplane. With
 iSBX® Multimodule™ board mounted,
 vertical height increases to 1.13 in.
 (2.87 cm)

Electrical Characteristics

Power Requirements

+ 5V DC @ 3 A
 + 10V DC @ TBD *Multimodule™
 + 12V DC @ 0.02 A *Multimodule™
 + 12V DC @ 0.5 A

Environmental Characteristics

Temperature—0 to 55°C (operating); -55°C to 85°C
 (non-operating)

Humidity—up to 90% relative humidity without
 condensation (operating); all conditions
 without condensation or frost (non-
 operating)

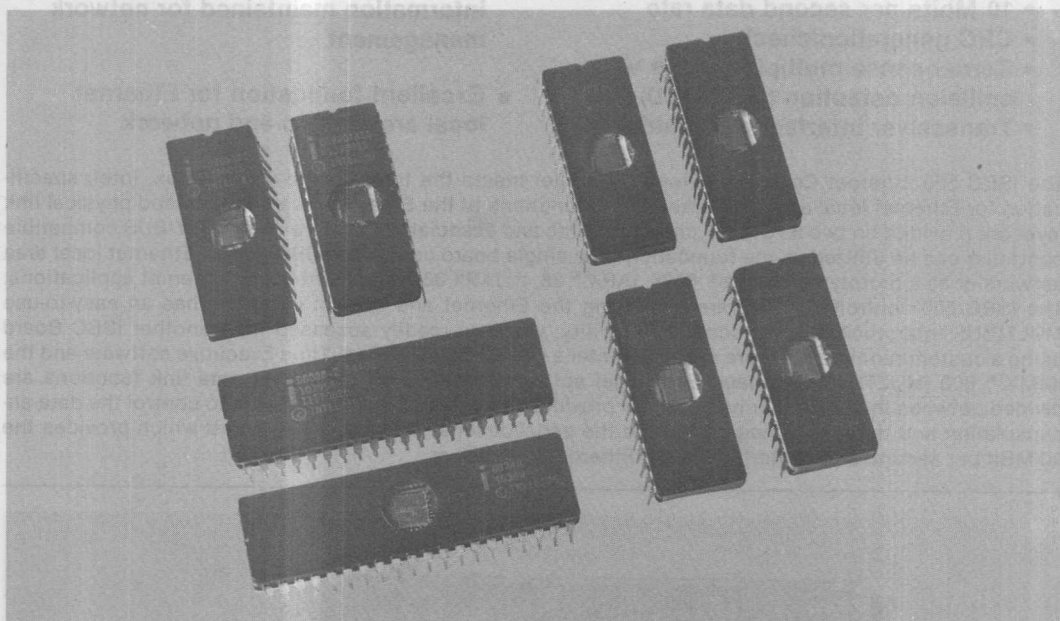
Reference Manual

Speech Transaction Design Manual (supplied)

iSBC® 577 SPEECH TRANSACTION RECOGNITION CHIP SET

- High-volume solution for speech I/O
- Fully compatible with iSBC 570, 576-generated software

The iSBC® 577 Speech Recognition Chip Set is a solution for your high-volume/maximum value-added speech I/O solution. The Chip Set contains the Intel-developed proprietary components from the iSBC® 576 Speech Transaction Board. With these components you can build the equivalent of the Speech Transaction Board into your own system. The Chip Set contains the digital front-end processors, a preprogrammed 8048 interface processor, the Speech Transaction Manager Firmware on 27128 EPROMs, and the 8086 microprocessor.



SPECIFICATIONS

Performance

—Refer to iSBC® 570 and iSBC® 576 performances.

Equipment Supplied

- 2—Preprogrammed 2920/21s (Digital Front-end Processor)
- 4—Preprogrammed 27128 (Speech Transaction Manager)
- 1—Preprogrammed 8048 (Interface Processor)
- 1—8086

ORDERING INFORMATION

Part Number Description

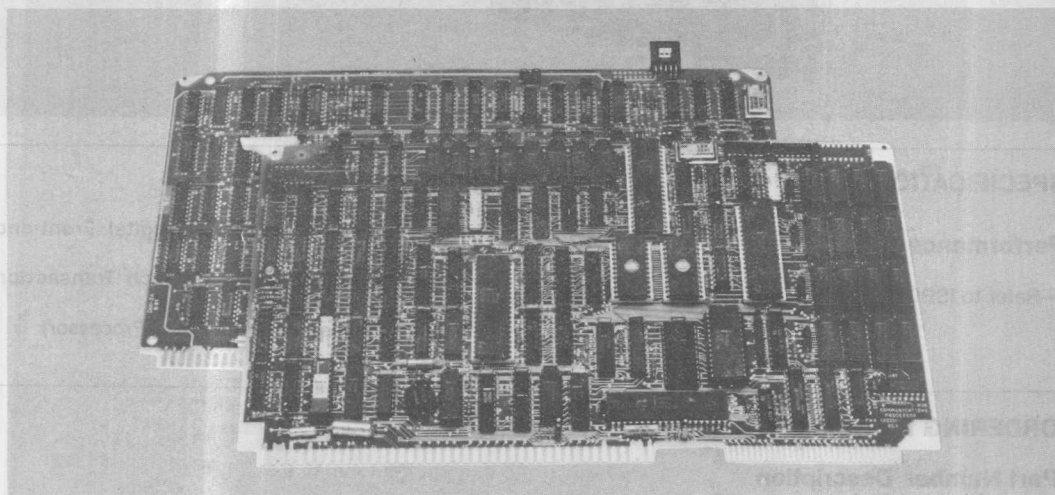
iSBC® 577	Speech Transaction Recognition Chip Set
-----------	---



iSBC™ 550 ETHERNET™ COMMUNICATIONS CONTROLLER

- Meets the version 1.0 tri-corporate Ethernet specification
- Ethernet data link layer support
 - Data encapsulation
 - Framing and packet control
 - Buffer management
- Ethernet physical link layer support
 - Serial/deserialization
 - 10 Mbits per second data rate
 - CRC generation/check
 - Carrier-sense multiple-access with collision detection (CSMA/CD)
 - Transceiver interface compatibility
- Easy-to-use MULTIBUS® inter-processor protocol supported in firmware
- Power-up confidence test assures integrity of on-board memory and programmable LSI
- Traffic, errors and collision information maintained for network management
- Excellent foundation for Ethernet local area end-to-end network

The iSBC 550 Ethernet Communications Controller meets the tri-corporate (DEC, Xerox, Intel) specification for Ethernet local area networks. All the functions of the Ethernet data link layer and physical link layer are provided on two 6.75 x 12" circuit boards and associated firmware. The MULTIBUS compatible controller can be utilized as the foundation for a single board computer (iSBC)-based Ethernet local area network or as a prototype for Intel® 8085, iAPX™ 88, or iAPX 86 component-based Ethernet applications. The iSBC 550 controller's firmware (supplying the Ethernet and system interface) has an easy-to-use MULTIBUS Interprocessor Protocol (MIP) facility, which is readily accessed from another iSBC Board using a custom run-time software system or Intel's iRMX™ 80/88/86 Real-Time Executive software and the iMMX™ 800 (MULTIBUS Message Exchange) software package. The Ethernet data link functions are divided between the processor board which provides the data link layer's software to control the data encapsulation and the link management, and the serial/deserialization (SerDes) board which provides the 10-MBit per second serial interface to the Ethernet transceiver.



The following are trademarks of Intel Corporation and may be used to describe Intel products: CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPL, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE, ICS, iAPX and iMMX. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

FUNCTIONAL DESCRIPTION

The iSBC 550 Ethernet Communications Controller is a two-board MULTIBUS-compatible set that offers high-speed Ethernet-compatible data transfer between digital devices operating at a 10-Mbit per sec data rate. The iSBC 550 controller can effectively support the needs of local area network applications, such as office automation, distributed data processing, factory data collection, research data collection, intelligent terminal and other EDP-related products.

Ethernet Specification

The Ethernet network is a local area network concept that is jointly being supported by Intel Corporation, Digital Equipment Corporation, and Xerox Corporation. The network is designed to link systems over a distance of up to 2500 meters using an available 50-ohm coaxial cable. Several hundred stations may be connected to the cable which supports a data rate of 10 Megabits per second. The data is encapsulated in a packet message format. The data signal is a base-band, Manchester-encoded type that is self-synchronizing.

The jointly developed Ethernet specification, "The Ethernet, A Local Area Network Data Link Layer and Physical Link Layer Specification, Version 1.0, September 30, 1980", precisely defines the two lower layers of a local area network architecture where the system is a series of independent layers. The lowest layer, the physical link layer, is concerned with coaxial cable interface. The data link layer supports the peer protocol's statistical contention resolution (CSMA/CD) and link management functions. All additional network layers are defined by the user during the implementation of the application-specific layers.

Ethernet Data Link Layer Support

The iSBC 550 processor board provides the data link layer's software to control the data encapsulation and the link management, including frame delimitation, address handling, error detection, and collision handling. After the iSBC 550 processor board is initialized upon system start-up or reset, the data link firmware is ready to service the local area network commands. An example of a command structure sent the iSBC controller to receive a packet of data from the Ethernet link is shown in Figure 1. The message passed via the MIP (MULTIBUS Interprocessor Protocol) interface is composed of two parts, the iSBC 550 controller information (including the command and

associated data), and the required Ethernet information.

ISBC 550 CONTROLLER INFORMATION	RESERVED DATA	(14 bytes)
	COMMAND	(1 byte)
	RESERVED DATA	(7 bytes)
ETHERNET INFORMATION	DESTINATION	(6 bytes)
	SOURCE	(6 bytes)
	TYPE	(2 bytes)
	DATA	(46-1500 bytes)

Figure 1. Data Link for SUPPLYBUF Command Format

Shown in Table 1 are eight external Ethernet controller commands available to a user's application via the MIP interface. The commands manage the Ethernet multicast address recognition, message type connection, message flow, and overall network statistics.

Table 1. External Controller Commands

Command	Function
CONNECT	Indicates the data link message TYPE to be <i>connected</i> to user program.
DISCONNECT	<i>Disconnects</i> the data link TYPE from the user's application.
ADDMCID	<i>Adds</i> a multicast ID for recognition.
DELETEMCID	<i>Delete</i> the specified multicast ID.
TRANSMIT	<i>Transmit</i> a data packet to the Ethernet link.
SUPPLYBUF	<i>Supplies</i> a buffer for packet reception from the Ethernet link ("receive" function).
READ	<i>Read</i> the statistical variables maintained by data link layer.
READC	<i>Read and clear</i> the statistical variables.

Ethernet Physical Link Layer Support

The Serialization/Deserialization (SerDes) board provides the required electrical characteristics of the physical link layer of the Ethernet architecture for a transceiver interface. The transceiver is a device physically attached to the coax cable which does signal conditioning for transmitting and receiving.

Many major functions are controlled by the SerDes board. These functions include serialization/deserialization, packet framing, Manchester encoding/decoding, transmit data flow control, receive data flow control, destination address decoding for received message, CRC generation and

checking, and diagnostics for CRC error, loop-back, transmit timeout, and CSMA/CD (Carrier-Sense Multiple-Access with Collision-Detection).

Easy-To-Use Interface

One of the iSBC 550 controller boards is an iAPX 88-based processor board which has firmware support for the user's application interface. The programmatic interface utilizes the MULTIBUS Interprocessor Protocol (MIP) interface to the processor board. This interface is concerned with the message-passing protocol between multiple-processors. The iMMX 800 (MULTIBUS Message Exchange) software supports the MIP interface and offers a convenient quick-start method for users of Intel's iRMX 80, iRMX 88 executives and iRMX 86 operating system products for an Ethernet-based application.

Confidence Test

An effective diagnostic function is implemented in firmware on the processor board. This function is invoked at system initialization during both power-up and system reset time. These functions include: packet CRC checking, memory test, controller loopback, and other error tests. The tests provide a fundamental level of controller integrity.

Network Statistics

Statistics maintained by the data link firmware include packet traffic counts, collision information

and error totals. This information can be effectively utilized by the user's application to understand the network's operation.

End-To-End Networking Foundation

The iSBC 550 controller provides the foundation data link layer and the physical link layer for a local area network architecture. Typically, the higher levels are user-defined and include the transport and the session control layers. The transport control layer is concerned with the end-to-end communications and the virtual channel connection via a port-to-port address. The session control layer provides the process-to-process control function which includes symbolic name binding and the establishment of the virtual connection via the transport control layer. In addition, the session control provides the specific error and recovery control responsible for message delivery.

The higher levels of the local area network architecture (see Figure 2) which use the data link layer are outside of the Ethernet standard, but can be implemented quickly on companion iSBC boards (e.g., iSBC 80/24, iSBC 88/25, iSBC 86/12A) running under the iRMX 80/88/86 Real-Time Multitasking Executives, respectively, and associated iMMX MULTIBUS Message Exchange (iMMX 800) software. Special iSBC 550 device driver software compatible with the iRMX 86 and iRMX 88 file systems is provided in the iMMX 800 package.

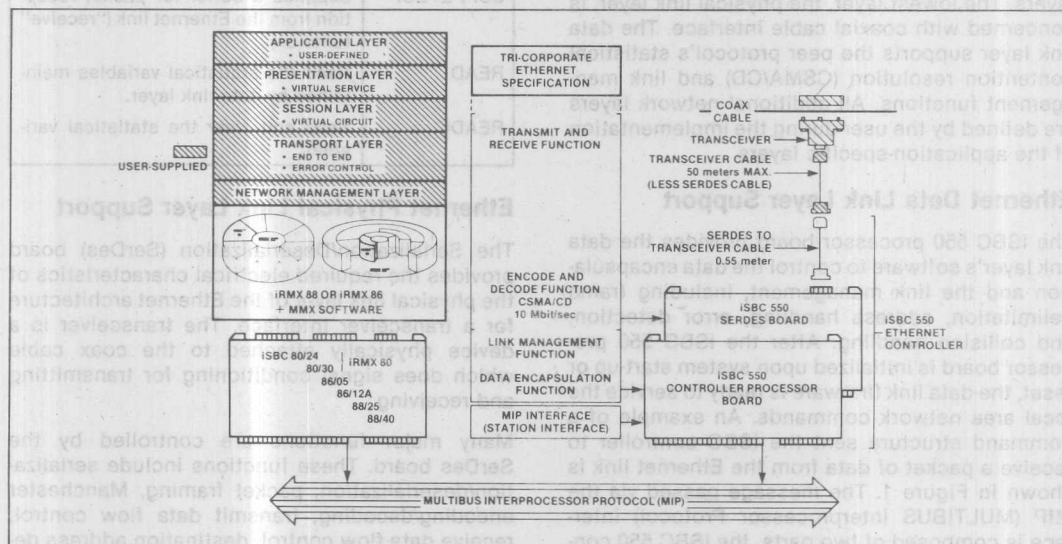


Figure 2. Ethernet Architecture and Implementation

SPECIFICATIONS

Memory Addressing Capability

MULTIBUS System Bus — (00000-EFFF)

Ethernet I/O Channels

One Ethernet electrically-compatible transceiver line on the SerDes board.

Interface Specifications

MULTIBUS System Bus — All signals TTL compatible.

Transceiver — All signals Ethernet specifications transceiver compatible.

Serial Communications Characteristics

Bit Serial Frame — Provides 64-bit *preamble*, 48-bit *destination* address, 48-bit *source* address, 16-bit *type*, 46-1500 bytes for *data*, and a *frame* check sequence of 32 bits.

Ethernet Network Specifications Supported

Coax Cable Length — 500-meter max.

Transceiver Cable Length — 50-meter max.

Number of Stations — 100 max.

Baud Rate — 10-Mbit/sec

System Clock

5.00 MHz, $\pm 0.1\%$

Physical Characteristics (Both Boards)

Width — 12.00 in. (30.48 cm) (each board)

Height — 6.75 in. (17.15 cm) (each board)

Depth — 0.5 in. (1.27 cm) (each board)

Weight — 3.5 lb (1.6 kg) (both boards)

SerDes to Transceiver Cable

Length — 0.55 meter (22 in.). Four pair twisted-wire cable with SerDes connector and transceiver interface connector.

Electrical Characteristics

Power requirements for both boards

+ 5 VDC @ 9.0A max.

+ 12 VDC @ 0.5A max.

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Relative Humidity — To 90% (without condensation)

Connectors

Interface	Pins (qty)	Centers (in.)	Mating Connectors
MULTIBUS System	86	0.156	Viking 2KH43/9AMK12
SerDes Edge Connector	10	0.1	AMP 87631-5 Housing AMP 87195-9 Pins
Transceiver	15	0.1	Cinch Type DA 51220-1

Reference Manuals

121746 — iSBC 550 Ethernet Communications Controller Hardware Reference Manual (NOT SUPPLIED)

121769 — The Ethernet Communications Controller Programmer's Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

ORDERING INFORMATION

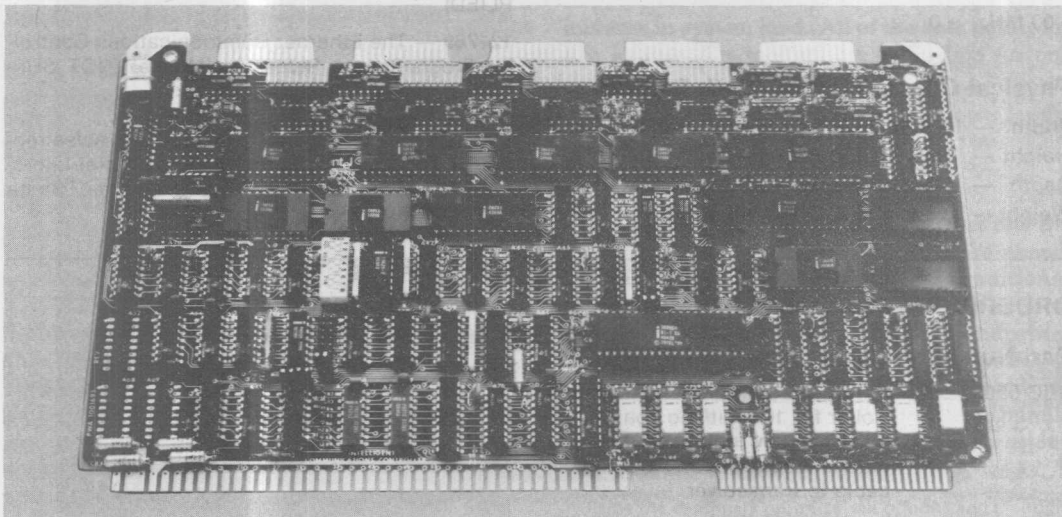
Part Number Description

SBC 550 Ethernet Communications Controller for 10 Mbit/sec coaxial transmission. Includes Ethernet data link control software and cable to transceiver.

iSBC 544 INTELLIGENT COMMUNICATIONS CONTROLLER

- iSBC Communications Controller acting as a single board communications computer or an intelligent slave for communications expansion
- On-board dedicated 8085A Micro-processor providing communications control and buffer management for four programmable synchronous/asynchronous channels
- Sockets for up to 8K bytes of EPROM
- 16K bytes of dual port dynamic read/write memory with on-board refresh
- Extended MULTIBUS addressing permits iSBC 544 board partitioning into 16K-byte segments in a 1-megabyte address space
- Ten programmable parallel I/O lines compatible with Bell 801 Automatic Calling Unit
- Twelve levels of programmable interrupt control
- Individual software programmable baud rate generation for each serial I/O channel
- Three independent programmable interval timer/counters
- Interface control for auto answer and auto originate modem

The iSBC 544 Intelligent Communications Controller is a member of Intel's family of single-board computers, memory, I/O, and peripheral controller boards. The iSBC 544 board is a complete communications controller on a single 6.75 x 12.00 inch printed circuit card. The on-board 8085A CPU may perform local communications processing by directly interfacing with on-board read/write memory, nonvolatile read only memory, four synchronous/asynchronous serial I/O ports, RS232/RS366 compatible parallel I/O, programmable timers, and programmable interrupts.



Intelligent Communications Controller

Two Mode Operation — The iSBC 544 board is capable of operating in one of two modes: 1) as a single board communications computer with all computer and communications interface hardware on a single board; 2) as an "intelligent bus slave" that can perform communications related tasks as a peripheral processor to one or more bus masters. The iSBC 544 may be configured to operate as a stand-alone single board communications computer with all MPU, memory and I/O elements on a single board. In this mode of operation, the iSBC 544 may also interface with expansion memory and I/O boards (but no additional bus masters). The iSBC 544 performs as an intelligent slave to the bus master by performing all communications related tasks. Complete synchronous and asynchronous I/O and data management are controlled by the on-board 8085A CPU

can be managed entirely on-board, freeing the bus master to perform other system functions.

Architecture — The iSBC 544 board is functionally partitioned into three major sections: I/O, central computer, and shared dual port RAM memory (Figure 1). The I/O hardware is centered around the four Intel 8251A USART devices providing fully programmable serial interfacing. Included here as well is a 10-bit parallel interface compatible with the Bell 801 automatic calling unit, or equivalent. The I/O is under full control of the on-board CPU and is protected from access by system bus masters. The second major segment of the intelligent communications controller is a central computer, with an 8085A CPU providing powerful processing capability. The 8085A together with on-board EPROM / ROM, static RAM, programmable timers/counters, and program-

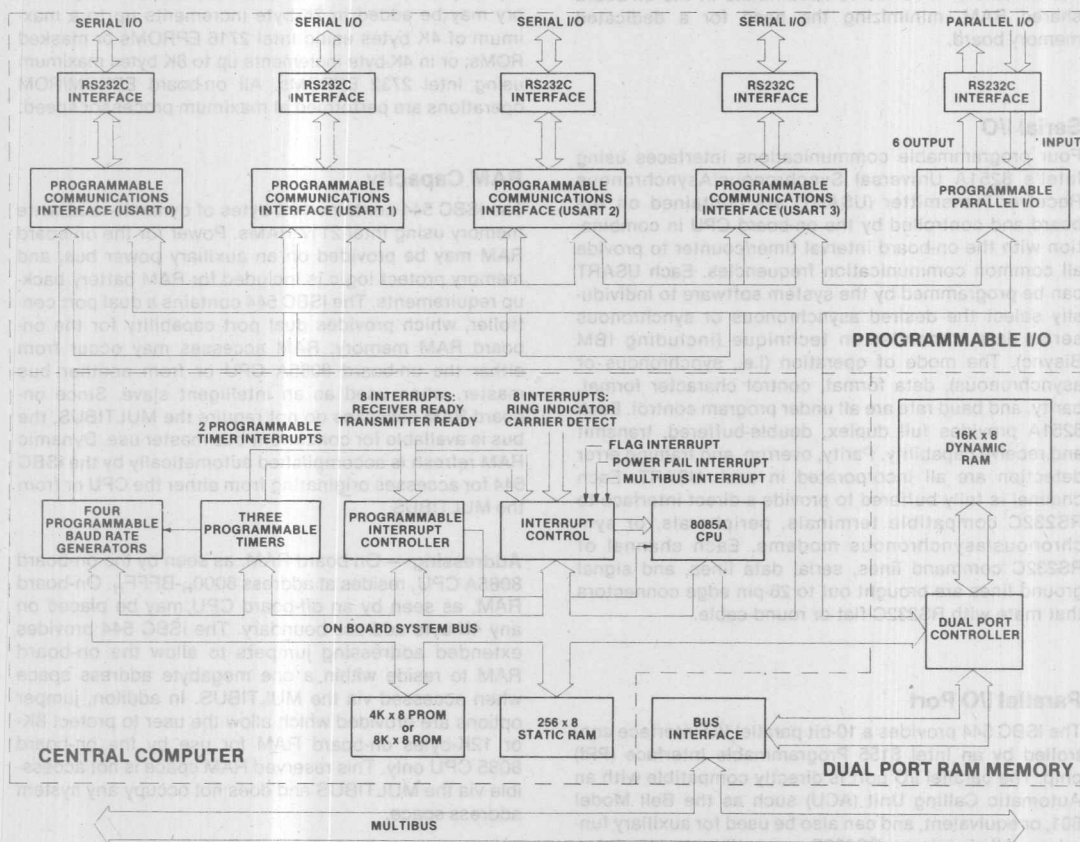


Figure 1. iSBC 544 Intelligent Communications Controller Block Diagram

mable interrupt control provide the intelligence to manage sophisticated communications operations on-board the iSBC 544 board. The timer/counters and interrupt control are also common to the I/O area providing programmable baud rates to the USARTs and prioritizing interrupts generated from the USARTs. The central computer functions are protected for access only by the on-board 8085A. Likewise, the on-board 8085A may not gain access to the system bus when being used as an intelligent slave. When the iSBC 544 is used as a bus master, the on-board 8085A CPU controls complete system operation accessing on-board functions as well as memory and I/O expansion. The third major segment, dual port RAM memory, is the key link between the iSBC 544 intelligent slave and bus masters managing the system functions. The dual port concept allows a common block of dynamic memory to be accessed by the on-board 8085A CPU and off-board bus masters. The system program can, therefore, utilize the shared dual port RAM to pass command and status information between the bus masters and on-board CPU. In addition, the dual port concept permits blocks of data transmitted or received to accumulate in the on-board shared RAM, minimizing the need for a dedicated memory board.

Serial I/O

Four programmable communications interfaces using Intel's 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) are contained on the board and controlled by the on-board CPU in combination with the on-board interval timer/counter to provide all common communication frequencies. Each USART can be programmed by the system software to individually select the desired asynchronous or synchronous serial data transmission technique (including IBM Bisync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. Each 8251A provides full duplex, double-buffered, transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in each USART. Each channel is fully buffered to provide a direct interface to RS232C compatible terminals, peripherals, or synchronous/asynchronous modems. Each channel of RS232C command lines, serial data lines, and signal ground lines are brought out to 26-pin edge connectors that mate with RS232C flat or round cable.

Parallel I/O Port

The iSBC 544 provides a 10-bit parallel I/O interface controlled by an Intel 8155 Programmable Interface (PPI) chip. The parallel I/O port is directly compatible with an Automatic Calling Unit (ACU) such as the Bell Model 801, or equivalent, and can also be used for auxiliary functions. All signals are RS232C compatible, and the interface cable signal assignments meet RS366 specifications. For systems not requiring an ACU interface, the parallel I/O port can be used for any general purpose interface requiring RS232C compatibility.

Central Processing Unit

Intel's powerful 8-bit n-channel 8085A CPU, fabricated on a single LSI chip, is the central processor for the iSBC 544. The 8085A CPU is directly software compatible with the Intel 8080A CPU. The 8085A contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators. The minimum instruction execution time is 1.45 microseconds. The 8085A CPU has a 16-bit program counter. An external stack, located within any portion of iSBC 544 read/write memory, may be used as a last-in/first-out storage area for the contents of the program counter, flags, accumulator, and all of the six general purpose registers. A 16-bit stack pointer controls the addressing of this external stack. This stack provides subroutine nesting bounded only by memory size.

EPROM/ROM Capacity

Sockets for up to 8K bytes of nonvolatile read only memory are provided on the iSBC 544 board. Read only memory may be added in 2K-byte increments up to a maximum of 4K bytes using Intel 2716 EPROMs or masked ROMs; or in 4K-byte increments up to 8K bytes maximum using Intel 2732 EPROMs. All on-board EPROM/ROM operations are performed at maximum processor speed.

RAM Capacity

The iSBC 544 contains 16K bytes of dynamic read/write memory using Intel 2117 RAMs. Power for the on-board RAM may be provided on an auxiliary power bus, and memory protect logic is included for RAM battery back-up requirements. The iSBC 544 contains a dual port controller, which provides dual port capability for the on-board RAM memory. RAM accesses may occur from either the on-board 8085A CPU or from another bus master, when used as an intelligent slave. Since on-board RAM accesses do not require the MULTIBUS, the bus is available for concurrent bus master use. Dynamic RAM refresh is accomplished automatically by the iSBC 544 for accesses originating from either the CPU or from the MULTIBUS.

Addressing — On board RAM, as seen by the on-board 8085A CPU, resides at address 8000_H-BFFF_H. On-board RAM, as seen by an off-board CPU, may be placed on any 4K-byte address boundary. The iSBC 544 provides extended addressing jumpers to allow the on-board RAM to reside within a one megabyte address space when accessed via the MULTIBUS. In addition, jumper options are provided which allow the user to protect 8K- or 12K-bytes on-board RAM for use by the on-board 8085 CPU only. This reserved RAM space is not accessible via the MULTIBUS and does not occupy any system address space.

Static RAM — The iSBC 544 board also has 256 bytes of static RAM located on the Intel 8155 PPI. This memory is only accessible to the on-board 8085A CPU and is located at address 7F00_H-7FFF_H.

Programmable Timers

The iSBC 544 board provides seven fully programmable and independent interval timer/counters utilizing two Intel 8253 Programmable Interval Timers (PIT), and the Intel 8155. The two Intel 8253 PITs provide six independent BCD or binary 16-bit interval timer/counters and the 8155 provides one 14-bit binary timer/counter. Four of the PIT timers (BDGO-3) are dedicated to the USARTs providing fully independent programmable baud rates.

Three General Use Timers — The fifth timer (BDG4) may be used as an auxiliary baud rate to any of the four USARTs or may alternatively be cascaded with timer six to provide extended interrupt intervals. The sixth PIT timer/counter (TINT1) can be used to generate interrupt intervals to the on-board 8085A. In addition to the timer/counters on the 8253 PITs, the iSBC 544 has a 14-bit timer available on the 8155 PPI providing a third general use timer/counter (TINT0). This timer output is jumper selectable to the interrupt structure of the on-board 8085A CPU to provide additional timer/counter capability.

Timer Functions — In utilizing the iSBC 544 board, the systems designer simply configures, via software, each timer independently to meet systems requirements. Whenever a given baud rate or interrupt interval is needed, software commands to the programmable timers select the desired function. The on-board PITs together with the 8155 provide a total of seven timer/counters and six operating modes. Mode 3 of the 8253 is the primary operating mode of the four dedicated USART baud rate generators. The timer/counters and useful modes of operation for the general use timer/counters are shown in Table 1.

Interrupt Capability

The iSBC 544 board provides interrupt service for up to 21 interrupt sources. Any of the 21 sources may interrupt the intelligent controller, and all are brought through the interrupt logic to 12 interrupt levels. Four interrupt levels are handled directly by the interrupt processing capability of the 8085A CPU and eight levels are serviced from an Intel 8259A Programmable Interrupt Controller (PIC) routing an interrupt request output to the INTR input of the 8085A (see Table 2).

Interrupt Sources — The 22 interrupt sources originate from both on-board communications functions and the Multibus. Two interrupts are routed from each of the four USARTs (8 interrupts total) to indicate that the transmitter and receiver are ready to move a data byte to or from the on-board CPU. The PIC is dedicated to accepting these 8 interrupts to optimize USART service request. One of eight interrupt request lines are jumper selectable for direct interface from a bus master via the system bus. Two auxiliary timers (TINT0 from 8155 and TINT1 from 8253) are jumper selectable to provide general purpose counter/timer interrupts. A jumper selectable Flag Interrupt is generated to allow any bus master to interrupt the iSBC 544 by writing into the base address of the shared dual port memory accessible to the system. The Flag Interrupt is then cleared by the iSBC 544 when the on-board processor reads the base address. This interrupt provides an interrupt link between

Table 1. Programmable Timer Functions

Function	Operation	Counter
Interrupt on Terminal Count (Mode 0)	When terminal count is reached, an interrupt request is generated. This function is useful for generation of real-time clocks.	8253 TINT1
Rate Generator (Mode 2)	Divide by N counter. The output will go low for one input clock cycle and high for N-1 input clock periods.	8253 BDG4 *
Square-Wave Rate Generator (Mode 3)	Output will remain high until one-half the TC has been completed, and go low for the other half of the count. This is the primary operating mode used for generating a Baud rate clocked to the USARTs.	8253 BDG0-4 TINT1
Software Triggered Strobe (Mode 4)	When the TC is loaded, the counter will begin. On TC the output will go low for one input clock period.	8253 BDG4 * TINT1
Single Pulse	Single pulse when TC reached.	8155 TINT0
Repetitive Single Pulse	Repetitive single pulse each time TC is reached until a new command is loaded.	8155 TINT0

* BDG4 is jumper selectable as an auxiliary baud rate generator to the USARTs or as a cascaded output to TINT1. BDG4 may be used in modes 2 and 4 only when configured as a cascaded output.

Table 2. Interrupt Vector Memory Locations

Interrupt Source	Vector Location	Interrupt Level
Power Fail	TRAP	24 _H 1
8253 TINT1	RST 7.5	3C _H 2
8155 TINT0		
Ring Indicator (1)	RST 6.5	34 _H 3
Carrier Detect		
Flag Interrupt	RST 5.5	2C _H 4
INT0-INT7 (1 of 8)		
RXRDY0	INTR	Program- mable 5-12
TXRDY0		
RXRDY1		
TXRDY1		
RXRDY2		
TXRDY2		
RXRDY3		
TXRDY3		

(1) Four ring indicator interrupts and four carrier detect interrupts are summed to the RST 6.5 input. The 8155 may be interrogated to inspect any one of the eight signals.

a bus master and intelligent slave (See System Programming). Eight inputs from the serial ports are monitored to detect a ring indicator and carrier detect from each of the four channels. These eight interrupt sources are summed to a single interrupt level of the 8085A CPU. If one of these eight interrupts occur, the 8155 PPI can then be interrogated to determine which port caused the interrupt. Finally, a jumper selectable Power Fail Interrupt is available from the Multibus to detect a power down condition.

8085 Interrupt — Thirteen of the twenty-two interrupt sources are available directly to four interrupt inputs of the on-board 8085A CPU. Requests routed to the 8085A interrupt inputs, TRAP, RST 7.5, RST 6.5 and RST 5.5 have a unique vector memory address. An 8085A jump instruction at each of these addresses then provides software linkage to interrupt service routines located independently anywhere in the Memory. All interrupt inputs with the exception of the TRAP may be masked via software.

8259A Interrupts — Eight interrupt sources signaling transmitter and receiver ready from the four USARTs are channeled directly to the Intel 8259A PIC. The PIC then provides vectoring for the next eight interrupt levels. Operating mode and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts transmitter and receiver interrupts from the four USARTs. It then determines which of the incoming requests is of highest priority, determines whether this request is of higher priority than the level currently being serviced, and, if appropriate, issues an interrupt to the CPU. The output of the PIC is applied directly to the INTR input of the 8085A. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. When the 8085A responds to a PIC interrupt, the PIC will generate a CALL instruction for each interrupt level. These addresses are equally spaced at intervals of 4 or 8 (software selectable) bytes. Interrupt response to the PIC is software programmable to a 32- or 64-byte block of memory. Interrupt sequences may be expanded from this block with a single 8085A jump instruction at each of these addresses.

Interrupt Output — In addition, the iSBC 544 board may be jumper selected to generate an interrupt from the on-board serial output data (SOD) of the 8085A. The SOD signal may be jumpered to any one of the 8 MULTIBUS interrupt lines (INT0-INT7) to provide an interrupt signal directly to a bus master.

Power-Fail Control

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the iSBC 635 Power Supply or equivalent.

Expansion Capabilities

When the iSBC 544 board is used as a single board communications controller, memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS™ compatible expansion boards. In this

mode, no other bus masters may be configured in the system. Memory may be expanded to a 65K byte capacity by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. Furthermore, multiple iSBC 544 boards may be included in an expanded system using one iSBC 544 board as a single board communications computer and additional controllers as intelligent slaves.

System Programming

In the system programming environment, the iSBC 544 board appears as an additional RAM memory module when used as an intelligent slave. The master CPU communicates with the iSBC 544 board as if it were just an extension of system memory. Because the iSBC 544 board is treated as memory by the system, the user is able to program into it a command structure which will allow the iSBC 544 board to control its own I/O and memory operation. To enhance the programming of the iSBC 544 board, the user has been given some specific tools. The tools are: 1) the flag interrupt, 2) an on-board RAM memory area that is accessible to both an off-board CPU and the on-board 8085A through which a communications path can exist, and 3) access to the bus interrupt line.

Flag Interrupt — The Flag Interrupt is generated anytime a write command is performed by an off-board CPU to the base address of the iSBC 544 board's RAM. This interrupt provides a means for the master CPU to notify the iSBC 544 board that it wishes to establish a communications sequence. In systems with more than one intelligent slave, the flag interrupt provides a unique interrupt to each slave outside the normal eight MULTIBUS interrupt lines (INT0-INT7).

On-Board RAM — The on-board 16K byte RAM area that is accessible to both an off-board CPU and the on-board 8085A can be located on any 4K boundary in the system. The selected base address of the iSBC 544 RAM will cause a flag interrupt when written into by an off-board CPU.

Bus Access — The third tool to improve system operation as an intelligent slave is access to the Multibus interrupt lines. The iSBC 544 board can both respond to interrupt signals from an off-board CPU, and generate an interrupt to the off-board CPU via the MULTIBUS.

System Development Capability

The development cycle of iSBC 544 board based products may be significantly reduced using the Intellec series microcomputer development systems. The Intellec resident macroassembler, text editor, and system monitor greatly simplify the design, development and debug of iSBC 544 system software. An optional ISIS-II diskette operating system provides a linker, object code locator, and library manager. A unique in-circuit emulator (ICE-85) option provides the capability of developing and debugging software directly on the iSBC 544 board.

SPECIFICATIONS

Serial Communications Characteristics

Synchronous — 5-8 bit characters; automatic sync insertion; parity.

Asynchronous — 5-8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection; break character detection.

Baud Rates

Frequency (KHz) ¹ (Software Selectable)	Baud Rate (Hz) ²	
	Synchronous	Asynchronous
153.6	—	+ 16 + 64
76.8	—	9600 2400
38.4	38400	4800 1200
19.2	19200	2400 600
9.6	9600	1200 300
4.8	4800	600 150
6.98	6980	300 75
		— 110

Notes:

- 1) Frequency selected by I/O writes of appropriate 16-bit frequency factor to Baud Rate Register.
- 2) Baud rates shown here are only a sample subset of possible software programmable rates available. Any frequency from 18.75 Hz to 614.4 KHz may be generated utilizing on-board crystal oscillator and 16-bit Programmable Interval Timer (used here as a frequency divider).

8085A CPU

Word Size — 8, 16 or 24 bits/instruction; 8 bits of data

Cycle Time — 1.45/usec \pm .1% for fastest executable instruction; i.e. four clock cycles.

Clock Rate — 2.76 MHz \pm .1%

System Access Time

Dual port memory — 740 nsec

Note: Assumes no refresh contention

Memory Capacity

On-Board ROM/PROM — 4K, or 8K bytes of user installed ROM or EPROM.

On-Board Static RAM — 256 bytes on 8155.

On-Board Dynamic RAM (on-board access) — 16K bytes. Integrity maintained during power failure with user-furnished batteries (optional).

On-Board Dynamic RAM (MULTIBUS access) — 4K, 8K, or 16K-bytes available to bus by switch selection.

Memory Addressing

On-Board ROM/PROM — 0-0FFF (using 2716 EPROMs or masked ROMs); 0-1FFF (using 2732A EPROMs)

On-Board Static Ram — 256 bytes: 7F00-7FFF

On-Board Dynamic RAM (on-board access) — 16K bytes: 8000-BFFF.

On-Board Dynamic RAM (MULTIBUS access) — any 4K increment 00000-FF000 which is switch and jumper selectable. 4K- 8K- or 16K-bytes can be made available to the bus by switch selection.

I/O Capacity

Serial — 4 programmable channels using four 8251A USARTs.

Parallel — 10 programmable lines available for Bell 801 ACU, or equivalent use. Two auxiliary jumper selectable signals.

I/O Addressing

On-Board Programmable I/O

Port	Data	Control
USART 0	D0	D1
USART 1	D2	D3
USART 2	D4	D5
USART 3	D6	D7
8155 PPI	E9 (Port A)	E8
	EA (Port B)	
	EB (Port C)	

Interrupts

Addresses for 8259A Registers (Hex notation, I/O address space)

E6 Interrupt request register

E6 In-service register

E7 Mask register

E6 Command register

E7 Block address register

E6 Status (polling register)

Note: Several registers have the same physical address: Sequence of access and one data bit of the control word determines which register will respond.

Interrupt levels routed to the 8085 CPU automatically vector the processor to unique memory locations:

24 TRAP

3C RST 7.5

34 RST 6.5

2C RST 5.5

Timers

Addresses for 8253 Registers (Hex notation, I/O address space)

Programmable Interrupt Timer One

D8	Timer 0	BDG0
D9	Timer 1	BDG1
DA	Timer 2	BDG2
DB	Control register	

Programmable Interrupt Timer Two

DC	Timer 0	BDG3
DD	Timer 1	BDG4
DE	Timer 2	TINT1
DF	Control register	

Address for 8155 Programmable Timer

E8	Control
ED	Timer (LSB) TINT0
EC	Timer (MSB) TINT0

Input frequencies — Jumper selectable reference 1.2288 MHz $\pm 1\%$ (.814 usec period nominal) or 1.843 MHz $\pm 1\%$ crystal (0.542 usec period, nominal)

Output Frequencies (at 1.2288 MHz)

Function	Single timer/counter		Dual timer/counter (two timers cascaded)	
	Min	Max	Min	Max
Real-time interrupt interval	1.63 usec	53.3 usec	3.26 usec	58.25 min
Rate Generator (frequency)	18.75 Hz	614.4 KHz	0.00029 Hz	307.2 KHz

Interfaces

Serial I/O — EIA Standard RS232C signals provided and supported:

Carrier Detect	Receive Data
Clear to Send	Ring Indicator
Data Set Ready	Secondary Receive Data *
Data Terminal Ready	Secondary Transmit Data *
Request to Send	Transmit Clock
Receive Clock	Transmit Data
	DTE Transmit Clock

* Optional if parallel I/O port is not used as Automatic Calling Unit.

Parallel I/O — Four inputs and eight outputs (includes two jumper selectable auxiliary outputs). All signals compatible with EIA Standard RS232C. Directly compatible with Bell Model 801 Automatic Calling Unit, or equivalent.

MULTIBUS — Compatible with ISBC MULTIBUS.

On-Board Addressing

All communications to the parallel and serial I/O ports, to the timers, and to the interrupt controller, are via read and write commands from the on-board 8085A CPU.

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM for systems requiring battery backup of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

Connectors

Interface	Pins (qty)	Centers (in.)	Mating Connectors
Bus	86	0.156	Viking 2KH43/9AMK12
Parallel I/O	50	0.1	3M 3415-000 or AMP 88083-1
Serial I/O	26	0.1	3M 3462-000 or AMP 88373-5

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during the system power-down sequences.

Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-state	50
Address	Tri-state	15
Commands	Tri-state	32

Note: Used as a master in the single board communications computer mode.

Physical Characteristics

Width:	30.48 cm (12.00 inches)
Depth:	17.15 cm (6.75 inches)
Thickness:	1.27 cm (0.50 inch)
Weight:	3.97 gm (14 ounces)

Electrical Characteristics

DC Power Requirements

Current Requirements				
Configuration	$V_{CC} = +5V$ $\pm 5\%$ (max)	$V_{DD} = \pm 12V$ $\pm 5\%$ (max)	$V_{BB} = -5V(3)$ $\pm 5\%$ (max)	$V_{AA} = -12V$ $\pm 5\%$ (max)
With 4K EPROM (using 2716)	$I_{CC} = 3.4$ max max	$I_{DD} = 350$ mA max	$I_{BB} = 5$ mA max	$I_{AA} = 200$ mA max
Without EPROM	3.3A max	350 mA max	5 mA max	200 mA max
RAM only (1)	390 mA max	176 mA max	5 mA max	—
RAM(2) refresh only	390 mA max	20 mA max	5 mA max	—

- Notes:**
1. For operational RAM only, for AUX power supply rating.
 2. For RAM refresh only. Used for battery backup requirements. No RAM accessed.
 3. V_{BB} is normally derived on-board from V_{AA} , eliminating the need for a V_{BB} supply. If it is desired to supply V_{BB} from the bus, the current requirement is as shown.

Environmental Characteristics

Operating Temperature: 0°C to 55°C (32°F to 131°F)

Relative Humidity: To 90% without condensation

Reference Manual

502160 — ISBC 544 Intelligent Communications Controller Board Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

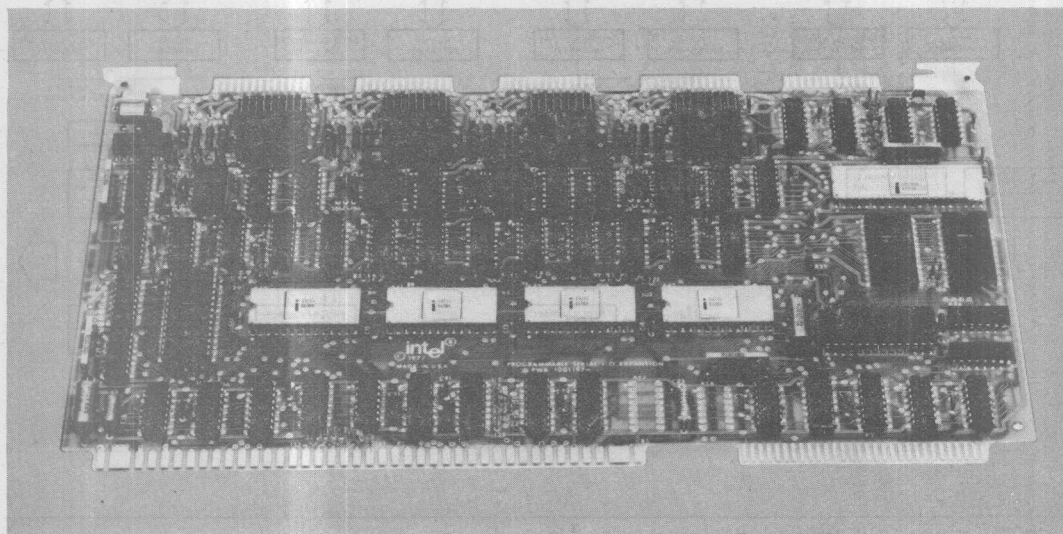
Part Number	Description
ISBC 544	Intelligent Communications Controller



iSBC™ 534 (or pSBC 534*) FOUR CHANNEL COMMUNICATION EXPANSION BOARD

- Serial I/O expansion through four programmable synchronous and asynchronous communications channels
- Individual software programmable baud rate generation for each serial I/O channel
- Two independent programmable 16-bit interval timers
- Sixteen maskable interrupt request lines with priority encoded and programmable interrupt algorithms
- Jumper selectable interface register addresses
- 16-bit parallel I/O interface compatible with Bell 801 automatic calling unit
- RS232C/CCITT V.24 interfaces plus 20 mA optically isolated current loop interfaces (sockets)
- Programmable digital loopback for diagnostics
- Interface control for auto answer and auto originate modems

The iSBC 534 Four Channel Communication Expansion Board is a member of Intel's complete line of memory and I/O expansion boards. The iSBC 534 interfaces directly to any single board computer via the MULTIBUS to provide expansion of system serial communications capability. Four fully programmable synchronous and asynchronous serial channels with RS232C buffering and provision for 20 mA optically isolated current loop buffering are provided. Baud rates, data formats, and interrupt priorities for each channel are individually software selectable. In addition to the extensive complement of EIA Standard RS232C signals provided, the iSBC 534 provides 16 lines of RS232C buffered programmable parallel I/O. This interface is configured to be directly compatible with the Bell Model 801 automatic calling unit. These capabilities provide a flexible and easy means for interfacing Intel iSBC based systems to RS232C and optically isolated current loop compatible terminals, cassettes, asynchronous and synchronous modems, and distributed processing networks.



*Same product, manufactured by Intel Puerto Rico, Inc.

FUNCTIONAL DESCRIPTION

Communications Interface

Four programmable communications interfaces using Intel's 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) are contained on the board.* Each USART can be programmed by the system software to individually select the desired asynchronous or synchronous serial data transmission technique (including IBM Bisync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. Each 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in each USART. Each set of RS232C command lines, serial data lines, and signal ground lines are brought out to 26-pin edge connectors that mate with RS232C flat or round cables.

16-Bit Interval Timers

The iSBC 534 provides six fully programmable and independent BCD and binary 16-bit interval timers utilizing two Intel 8253 programmable interval timers.* Four timers are available to the systems designer to generate baud rates for the USARTs under software control. Routing for the outputs from the other two counters is jumper selectable. Each may be independently routed to the programmable interrupt controller to provide real time clocking or to the USARTs (for applications requiring different transmit and receive baud rates). In utilizing the iSBC 534, the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given baud rate or

time delay is needed, software commands to the programmable timers select the desired function. Three functions of these timers are supported on the iSBC 534, as shown in Table 1. The contents of each counter may be read at any time during system operation.

Table 1. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached an interrupt request is generated. This function is used for the generation of real-time clocks.
Rate generator	Divide by N counter. The output will go low for one input clock cycle and high for N - 1 input clock periods.
Square wave rate generator	Output will remain high for one-half the count and low for the other half of the count.

Interrupt Request Lines

Two independent Intel 8259A programmable interrupt controllers (PIC's) provide vectoring for 16 interrupt levels.* As shown in Table 2, a selection of three priority processing algorithms is available to the system designer. The manner in which requests are serviced may thus be configured to match system requirements. Priority assignments may be reconfigured dynamically via software at any time during system operation. Any combination of interrupt levels may be masked through storage, via software, of a single byte to the interrupt mask register of each PIC. Each PIC's interrupt request

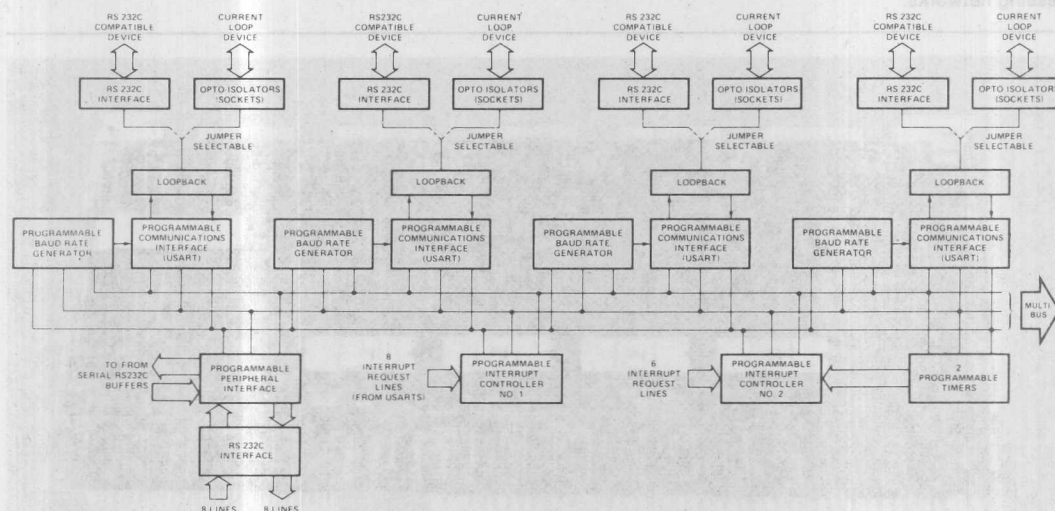


Figure 1. iSBC 534 Four Channel Communications Expansion Board Block Diagram

output line may be jumper selected to drive any of the nine interrupt lines on the MULTIBUS.

Table 2. Interrupt Priority Options

Algorithm	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.

Interrupt Request Generation — As shown in Table 3, interrupt requests may originate from 16 sources. Two jumper selectable interrupt requests (8 total) can be automatically generated by each USART when a character is ready to be transferred to the MULTIBUS system bus (i.e., receive buffer is full) or a character has been transmitted (transmit buffer is empty). Jumper selectable requests can be generated by two of the programmable timers (PITs), and six lines are routed directly from peripherals to accept carrier detect (4 lines), ring indicator, and the Bell 801 present next digit request lines.

Systems Compatibility

The iSBC 534 provides 16 RS232C buffered parallel I/O lines implemented utilizing an Intel 8255A program-

Table 3. Interrupt Assignments

Interrupt Request Line	PIC 0	PIC 1
0	PORT 0 R _x RDY	PIT 1 counter 1
1	PORT 0 T _x RDY	PIT 2 counter 2
2	PORT 1 R _x RDY	Ring indicator (all ports)
3	PORT 1 T _x RDY	Present next digit
4	PORT 2 R _x RDY	Carrier detect port 0
5	PORT 2 T _x RDY	Carrier detect port 1
6	PORT 3 R _x RDY	Carrier detect port 2
7	PORT 3 T _x RDY	Carrier detect port 3

mable peripheral interface (PPI) configured to operate in mode 0.* These lines are configured to be directly compatible with the Bell 801 automatic calling unit (ACU). This capability allows the iSBC 534 to interface to Bell 801 type ACUs and up to four modems or other serial communications devices. For systems not requiring interface to an ACU, the parallel I/O lines may also be used as general purpose RS232C compatible control lines in system implementation.

* Complete operational details on the Intel 8251A USART, the Intel 8253 Programmable Interval Timer, the Intel 8255A Programmable Peripheral Interface, and the Intel 8259A Programmable Interrupt Controller are contained in the Intel Component Data Catalog.

SPECIFICATIONS

Serial Communications Characteristics

Synchronous — 5-8 bit characters; internal or external character synchronization; automatic sync insertion.

Asynchronous — 5-8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection.

Sample Baud Rates¹

Frequency ² (kHz, Software Selectable)	Baud Rate (Hz)	
	Synchronous	Asynchronous
153.6	—	÷ 16 ÷ 64
76.8	—	9600 2400
38.4	38400	4800 1200
19.2	19200	2400 600
9.6	9600	1200 300
4.8	4800	600 150
6.98	6980	300 75
		— 110

Notes:

1. Baud rates shown here are only a sample subset of possible software-programmable rates available. Any frequency from 18.75 Hz to 614.4 kHz may be generated utilizing on-board crystal oscillator and 16-bit programmable interval timer (used here as frequency divider).

2. Frequency selected by I/O writes of appropriate 16-bit frequency factor to Baud Rate Register.

Interval Timer and Baud Rate Generator Frequencies

Input Frequency (On-Board Crystal Oscillator) — 1.2288 MHz ± 0.1% (0.813 µs period, nominal)

Function	Single Timer		Dual/Timer Counter (Two Timers Cascaded)	
	Min	Max	Min	Max
Real-Time Interrupt Interval	1.63 µs	53.3 ms	3.26 µs	58.25 minutes
Rate Generator (Frequency)	18.75 Hz	614.4 kHz	0.0029 Hz	307.2 kHz

Interfaces — RS232C Interfaces

EIA Standard RS232C Signals provided and supported:

Carrier detect	Receive data
Clear to send	Ring indicator
Data set ready	Secondary receive data
Data terminal ready	Secondary transmit data
Request to send	Transmit clock
Receive clock	Transmit data

Parallel I/O — 8 input lines, 8 output lines, all signals RS232C compatible

Bus — All signals MULTIBUS system bus compatible

I/O Addressing

The USART, interval timer, interrupt controller, and parallel interface registers of the iSBC 534 are configured as a block of 16 I/O address locations. The location of this block is jumper selectable to begin at any 16-byte I/O address boundary (i.e., 00H, 10H, 20H, etc.).

I/O Access Time

400 ns	USART registers
400 ns	Parallel I/O registers
400 ns	Interval timer registers
400 ns	Interrupt controller registers

Compatible Connectors/Cable

Interface	Pins (qty)	Centers (in.)	Mating Connectors	Cable
Bus	86	0.156	Viking 2KH43/9AMK12	N/A
Serial and parallel I/O	26	0.1	3M 3462-0001 or TI H312113	Intel iSBC 955

Compatible Opto-Isolators

Function	Supplier	Part Number
Driver	Fairchild General Electric Monsanto	4N33
Receiver	Fairchild General Electric Monsanto	4N37

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.50 in. (1.27 cm)

Weight — 14 oz (398 gm)

Electrical Characteristics

Average DC Current

Voltage	Without Opto-Isolators	With Opto-Isolators ¹
V _{CC} = +5V	1.9 A, max	1.9 A, max
V _{DD} = +12V	275 mA, max	420 mA, max
V _{AA} = -12V	250 mA, max	400 mA, max

Note

1. With four 4N33 and four 4N37 opto-isolator packages installed in sockets provided to implement four 20 mA current loop interfaces.

Environmental Characteristics

Operating Temperature — 0°C to +55°C

Reference Manual

502140-002 — iSBC 534 Hardware Reference Manual
(NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 534	Four Channel Communication Expansion Board



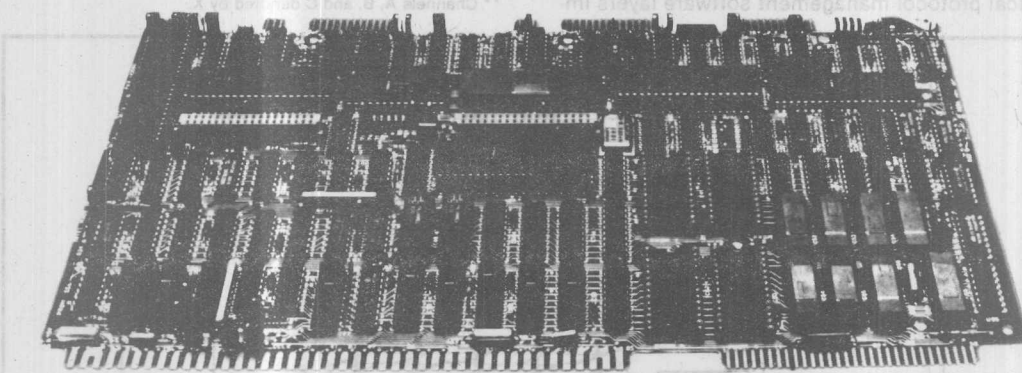
iSBC® 88/45



iSBC® 88/45 ADVANCED DATA COMMUNICATIONS PROCESSOR BOARD

- Three HDLC/SDLC half/full-duplex communication channels — optional ASYNC/SYNC on two channels
- Supports RS232C (including modem support), CCITT V.24, or RS422A/449 interfaces
- On-board DMA supports 800K baud operation
- Self-clocking NRZI SDLC loop data link interface
 - point-to-point
 - multidrop
- Software programmable baud rate generation
- iAPX 88/10 (8088-2) Microprocessor operates at 8 MHz
- iSBC® 337 Numeric Data Processor option supported
- 16K bytes static RAM (12K bytes dual-ported)
- Four 28-pin JEDEC sites for EPROM/RAM expansion; four additional 28-pin JEDEC sites added with iSBC® 341 board
- Two iSBX™ bus connectors
- MULTIBUS® interface supports Multimaster configuration

The iSBC 88/45 Advanced Data Communications Processor (ADCP) Board adds 8 MHz, iAPX 88/10 (8088-2) 8-bit microprocessor-based communications flexibility to the Intel line of OEM microcomputer systems. The iSBC 88/45 ADCP board offers asynchronous, synchronous, SDLC, and HDLC serial interfaces for gateway networking or general purpose solutions. The iSBC 88/45 ADCP board provides the CPU, system clock, EPROM/RAM, serial I/O ports, priority interrupt logic, and programmable timers to facilitate higher-level application solutions.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

INTEL CORPORATION, 1982

FUNCTIONAL DESCRIPTION

Three Communication Channels

Three programmable HDLC/SDLC serial interfaces are provided on the iSBC 88/45 ADCP board. The SDLC interface is familiar to IBM system and terminal equipment users. The HDLC interface is known by users of CCITT's X.25 packet switching interface.

One channel utilizes an Intel 8273 controller to manage the serial data transfers. Accepting the 8-bit data bytes from the local bus, the 8273 controller translates the data into the HDLC/SDLC format. The channel operates in half/full-duplex mode.

In addition to the synchronous mode, the 8273 controller operates asynchronously with NRZI encoded data which is found in systems such as the IBM 3650 Retail Store System. An SDLC loop configuration using iSBX 352 and iSBC 88/45 products is shown in Figure 1.

The two additional channels utilize the Intel 8274 Multi-Protocol Serial Controller (MPSC). The MPSC provides two independent half/full-duplex serial channels which provide asynchronous, synchronous, HDLC or SDLC protocol operations. The sync and async protocol operations are commonly used to communicate with inexpensive terminals and systems.

The three serial channels of the iSBC 88/45 ADCP board offer communications capability to manage a gateway application. The gateway application, as shown in Figure 1, manages diverse protocol requirements for data movement between channels. Typical protocol management software layers im-

plemented by the user include SNA terminal interfaces to IBM systems.

On-Board DMA

For high-speed communications, one MPSC channel has a DMA capacity to support an 800K baud rate. The second channel attached to the MPSC is capable of simultaneous 800K baud operation when configured with DMA capability, but is connected to an RS232C interface which is defined as 20K baud maximum. Figure 2 shows an RS422A/449 multidrop application which supports high-speed operation.

Interfaces Supported

The iSBC 88/45 ADCP board provides an excellent foundation to support these electrical and diverse software drivers protocol interfaces. The control lines, serial data lines, and signal ground lines are brought out to the three double-edge connectors. Figure 3 shows the cable to connector construction. Two connectors are pre-configured for RS422A/449. All three channels are configurable for RS232C/CCITT V.24 interfaces as shown in Table 1.

Table 1. iSBC® 88/45 Supported Configurations

Connection	Synchronous		Asynchronous	
	Modem	Direct	Modem*	Direct
point-to-point	X**	X	X	X
multidrop	X	X	X	X
loop	N.A.	N.A.	C (only)	C (only)

* Modem should not respond to break.

** Channels A, B, and C denoted by X.

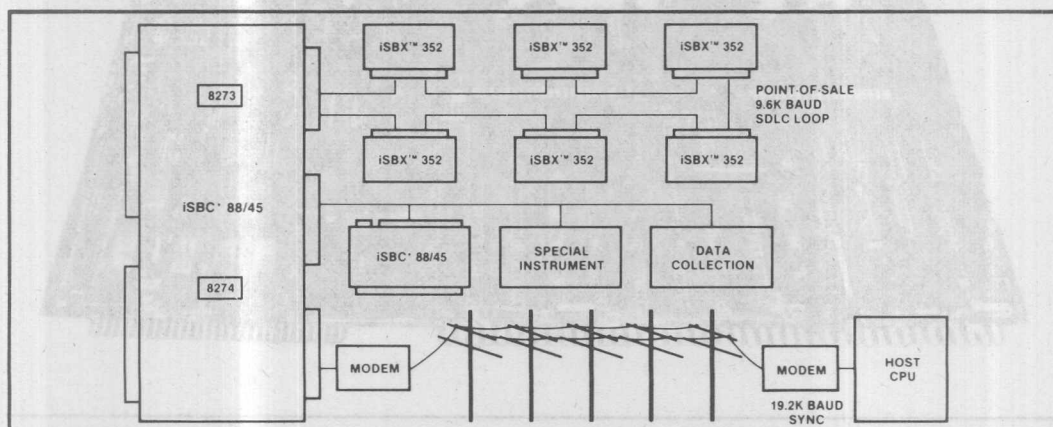


Figure 1. iSBC® 88/45 Gateway Processor Example

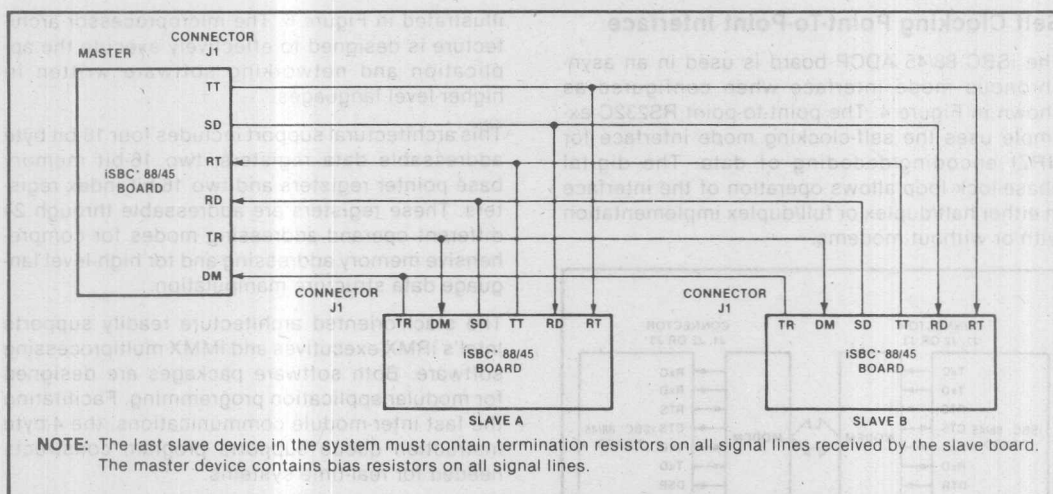


Figure 2. Synchronous Multidrop Network Configuration Example - RS422A

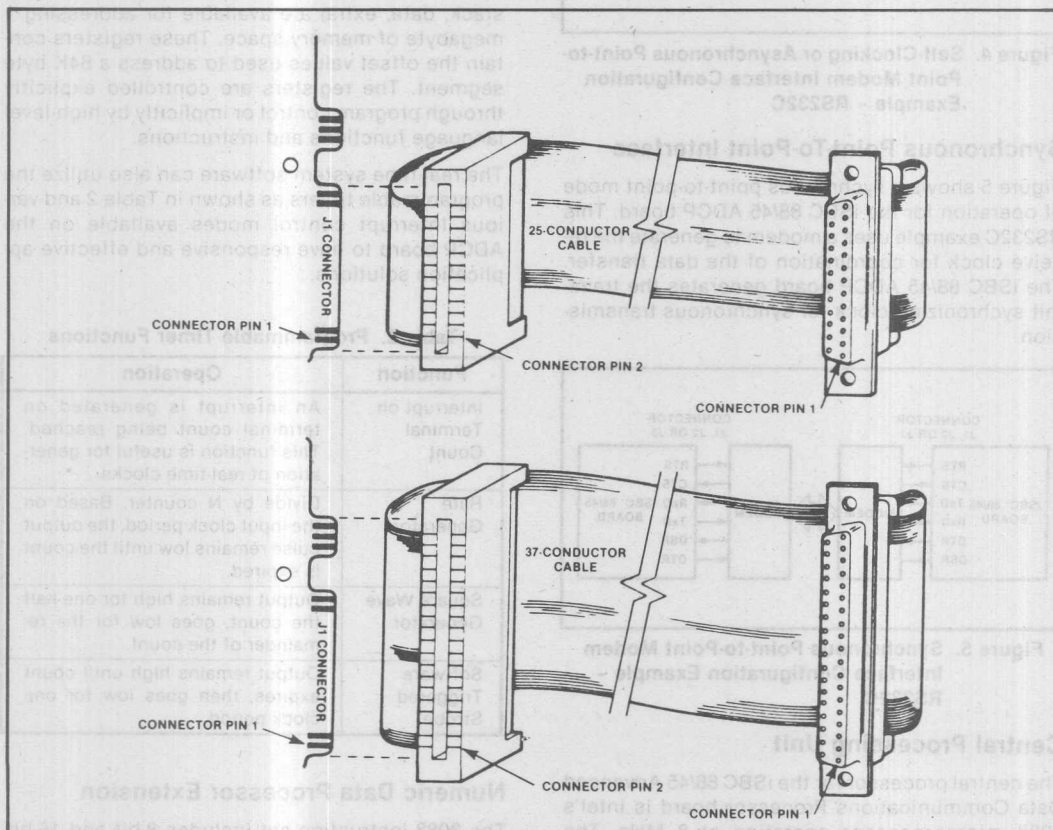


Figure 3. Cable Construction and Installation for RS232C and RS422A/449 Interface

Self Clocking Point-To-Point Interface

The iSBC 88/45 ADCP board is used in an asynchronous mode interface when configured as shown in Figure 4. The point-to-point RS232C example uses the self-clocking mode interface for NRZI encoding/decoding of data. The digital phase-lock loop allows operation of the interface in either half/duplex or full/duplex implementation with or without modems.

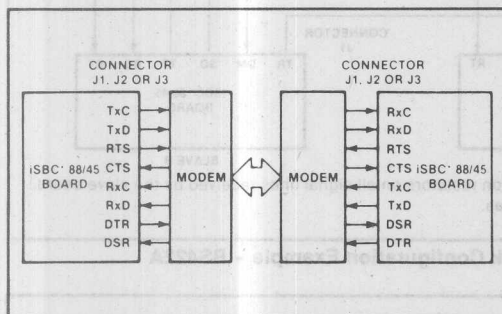


Figure 4. Self-Clocking or Asynchronous Point-to-Point Modem Interface Configuration Example - RS232C

Synchronous Point-To-Point Interface

Figure 5 shows a synchronous point-to-point mode of operation for the iSBC 88/45 ADCP board. This RS232C example uses a modem to generate the receive clock for coordination of the data transfer. The iSBC 88/45 ADCP board generates the transmit synchronizing clock for synchronous transmission.

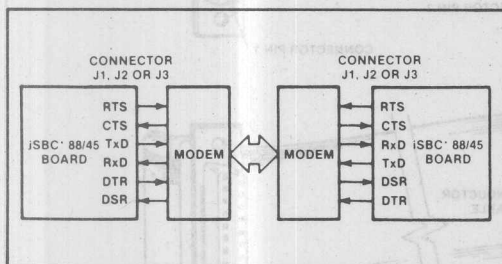


Figure 5. Synchronous Point-to-Point Modem Interface Configuration Example - RS232C

Central Processing Unit

The central processor for the iSBC 88/45 Advanced Data Communications Processor board is Intel's 8088 microprocessor operating at 8 MHz. The microprocessor interface to other functions is

illustrated in Figure 6. The microprocessor architecture is designed to effectively execute the application and networking software written in higher-level languages.

This architectural support includes four 16-bit byte addressable data registers, two 16-bit memory base pointer registers and two 16-bit index registers. These registers are addressable through 24 different operand addressing modes for comprehensive memory addressing and for high-level language data structure manipulation.

The stack-oriented architecture readily supports Intel's iRMX executives and iMMX multiprocessing software. Both software packages are designed for modular application programming. Facilitating the fast inter-module communications, the 4-byte instruction queue supports program constructs needed for real-time systems.

Since programs are segmented between pure procedure and data, four segment registers (code, stack, data, extra) are available for addressing 1 megabyte of memory space. These registers contain the offset values used to address a 64K byte segment. The registers are controlled explicitly through program control or implicitly by high-level language functions and instructions.

The real-time system software can also utilize the programmable timers as shown in Table 2 and various interrupt control modes available on the ADCP board to have responsive and effective application solutions.

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on Terminal Count	An interrupt is generated on terminal count being reached. This function is useful for generation of real-time clocks.
Rate Generator	Divide by N counter. Based on the input clock period, the output pulse remains low until the count is expired.
Square Wave Generator	Output remains high for one-half the count, goes low for the remainder of the count.
Software Triggered Strobe	Output remains high until count expires, then goes low for one clock period.

Numeric Data Processor Extension

The 8088 instruction set includes 8-bit and 16-bit signed and unsigned arithmetic operators for bi-

nary, BCD, and unpacked ASCII data. For enhanced numerics processing capability, the iSBC 337 MULTIMODULE Numeric Data Processor extends the 8088 architecture and data set¹.

The extended numerics capability includes over 60 numeric instructions offering arithmetic, trigonometric, transcendental, logarithmic, and exponential instructions. Many math-oriented applications utilize the 16-, 32-, and 64-bit integer, 32- and 64-bit floating point, 18-digit packed BCD, and 80-bit temporary data types.

16K Bytes Static Ram

The iSBC 88/45 ADCP board contains 16K bytes of high-speed static RAM, with 12K bytes dual-ported which is addressable from other MULTIBUS devices. When coupled with the high-speed DMA capability of the iSBC 88/45 ADCP board, the dual-ported memory provides effective data communication buffers. The dual-ported memory is useful for interprocessor message transfers.

¹ The iSBC 337 board requires the iSBC 88/45 ADCP board be jumpered to provide 4 MHz operation.

Interrupt Capability

The iSBC 88/45 ADCP board provides nine vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line. The additional eight interrupt levels are vectored via the Intel 8259A Programmable Interrupt Controller (PIC). As shown in Table 3, four priority processing modes are available to match interrupt servicing requirements. These modes and priority assignments are dynamically configurable by the system software.

Table 3. Programmable Interrupt Modes

Mode	Operation
Nested	Interrupt request line priorities fixed; interrupt 0 is the highest and 7 is the lowest.
Auto-Rotating	The interrupt priority rotates; once an interrupt is serviced it becomes the lowest priority.
Specific Priority	System software assigns lowest level priority. The other levels are sequenced based on the level assigned.
Polled	System software examines priority interrupt via interrupt status register.

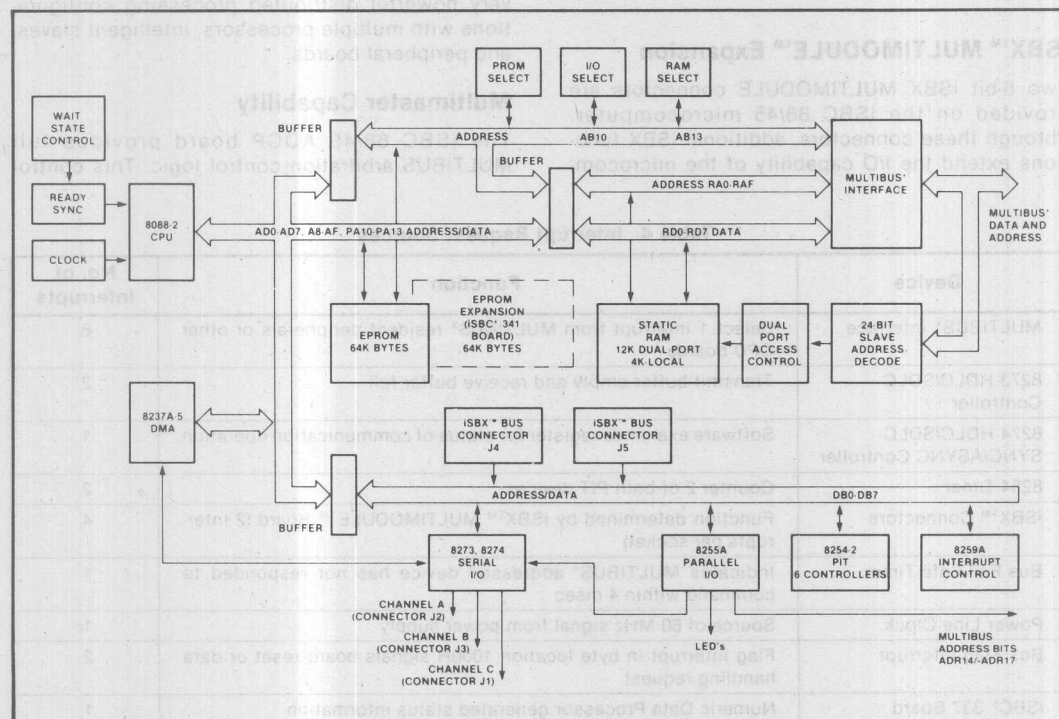


Figure 6. Block Diagram of the iSBC® 88/45 ADCP Board

Interrupt Request Generation

Listed in Table 4 are the devices and functions supported by interrupts on the iSBC 88/45 ADCP board. All interrupt signals are brought to the interrupt jumper matrix. Any of the 23 interrupt sources are strapped to the appropriate 8259A PIC request level. The PIC resolves requests according to the software selected mode and, if the interrupt is unmasked, issues an interrupt to the CPU.

EPROM/RAM Expansion

In addition to the on-board RAM, the iSBC 88/45 ADCP board provides four 28-pin JEDEC sockets for EPROM expansion. By using 2764 EPROMs, the board has 32K bytes of program storage. Three of the JEDEC standard sockets also support byte-wide static RAMs; using 8K x 8 static RAMs provides an additional 24K bytes of RAM.

Inserting the optional iSBC 341 MULTIMODULE EPROM expansion board onto the iSBC 88/45 ADCP board provides four additional 28-pin JEDEC sites. This expansion doubles the available program storage or extends the RAM capability by 32K bytes.

iSBX™ MULTIMODULE™ Expansion

Two 8-bit iSBX MULTIMODULE connectors are provided on the iSBC 88/45 microcomputer. Through these connectors, additional iSBX functions extend the I/O capability of the microcom-

puter. The iSBX connectors provide the necessary signals to interface to the local bus.

In addition to specialized or custom designed iSBX boards, the customer has a broad range of Intel iSBC MULTIMODULEs available, including parallel I/O, analog I/O, IEEE 488 GPIB, floppy disk, magnetic bubbles, video, and serial I/O boards.

The serial I/O MULTIMODULE boards include the iSBX 351 (one ASYNC/SYNC serial channel) and the iSBX 352 (one HDLC/SDLC serial channel) boards. Adding two iSBX 352 MULTIMODULE boards to the iSBC 88/45 ADCP provides a total of five HDLC/SDLC channels.

MULTIBUS® Multimaster Capabilities

OVERVIEW

The MULTIBUS system is Intel's industry standard microcomputer bus structure. Both 8- and 16-bit single board computers are supported on the MULTIBUS structure with 24 address and 16 data lines. In addition to expanding functions contained on a single board computer (e.g., memory and digital I/O), the MULTIBUS structure allows very powerful distributed processing configurations with multiple processors, intelligent slaves, and peripheral boards.

Multimaster Capability

The iSBC 88/45 ADCP board provides full MULTIBUS arbitration control logic. This control

Table 4. Interrupt Request Sources

Device	Function	No. of Interrupts
MULTIBUS® Interface	Select 1 interrupt from MULTIBUS® resident peripherals or other CPU boards	8
8273 HDLC/SDLC Controller	Transmit buffer empty and receive buffer full	2
8274 HDLC/SDLC SYNC/ASYNC Controller	Software examines register for status of communication operation	1
8254-Timer	Counter 2 of both PIT devices	2
iSBX™ Connectors	Function determined by iSBX™ MULTIMODULE™ Board (2 interrupts per socket)	4
Bus Fail Safe Timer	Indicates MULTIBUS® addressed device has not responded to command within 4 msec	1
Power Line Clock	Source of 60 MHz signal from power supply	1
Bus Flag Interrupt	Flag interrupt in byte location 1000H signals board reset or data handling request	2
iSBC® 337 Board	Numeric Data Processor generated status information	1
8237A-5	Signals end of 8237 DMA operation	1

logic allows up to three iSBC 88/45 ADCP boards or other bus masters, including iSBC 80 and iSBC 86 family boards to share the system bus using a serial (daisy chain) priority scheme. By using an external parallel priority decoder, the MULTIBUS system bus could be shared among sixteen masters.

The Intel standard MULTIBUS Interprocessor Protocol (MIP) software, implemented as the Intel iMMX 800 package for iRMX 80, iRMX 86, and iRMX 88 Real-Time Executives, fully supports multiple 8-and 16-bit distributed processor functions. The software manages the message passing protocol between microprocessors.

System Development Capabilities

The application development cycle for an iSBC 88/45 ADCP board is reduced and simplified through the usage of several Intel tools. The tools include the Intellec Series Microcomputer Development System, the ICE-88 In-Circuit Emulator, the iSBC 957B debug monitor software, and the iRMX 86 and iRMX 88 run-time support packages.

The Intellec Series Microcomputer Development System offers a complete development environment for the iSBC 88/45 software. In addition to the operating system, assembler, utilities and application debugger features provided with the system, the user optionally can utilize higher-level languages like PL/M, PASCAL, and FORTRAN.

The ICE-88 In-Circuit Emulator provides a link between the Intellec system and the target iSBC 88/45-based system for code loading and execution. The ICE-88 package assists the developer with the debugging and system integrating processes.

Run-Time Building Blocks

Intel offers run-time foundation software to support applications which range from general purpose to high-performance solutions. The iRMX 88 Real-time Multitasking Executive provides a multitasking structure which includes task scheduling, task management, intertask communications, and interrupt servicing for high-performance applications. The highly configurable modules make the system tailoring job easier whether one uses the compact executive or the complete executive with its variety of peripheral devices supported.

The iRMX 86 Operating System provides a very rich set of features and options to support sophisticated applications solutions. In addition to supporting real-time requirements, the iRMX 86 Operating System has a powerful, but easy-to-use human interface. When added to the sophisticated I/O system, the iRMX 86 Operating System is readily extended to support assembler, PL/M, PASCAL, and FORTRAN software development environments. The modular building block software lends itself well to customized application solutions.

SPECIFICATIONS

Word Size

Instruction — 8, 16, 24, or 32 bits

Data — 8 or 16 bits

System Clock

8 MHz — $\pm 0.1\%$

NOTE: Jumper selectable for 4 MHz operation with iSBC 337 Numeric Data Processor module or ICE-88 product.

Cycle Time

Basic Instruction Cycle at 8.00 MHz — 1.25 μ sec, 250 nsec (assumes instruction in the queue)

NOTE: Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles).

Memory Cycle Time

RAM — 500 nsec (no wait states)

EPROM — jumper selectable from 500 nsec to 625 nsec.

On-Board RAM* —

K Bytes	Hex Address Range
16 (total)	0000-3FFF
12 (dual-ported)	1000-3FFF

* Four iSBC 88/45 EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs (3 sockets); iSBC 341 (4 sockets)

Environmental Characteristics

Temperature — 0-55°C, free moving air across the base board and MULTIMODULE board

Humidity — 90%, non-condensing

Physical Characteristics

Width — 30.48 cm (12.00 in)

Length — 17.15 cm (6.75 in)

Height — 1.50 cm (0.59 in)

Weight — 6.20 gm (22 oz)

Memory Capacity/Addressing

On-Board EPROM* —

Device	Total K Bytes	Hex Address Range
2716	8	FE000-FFFF
2732A	16	FC000-FFFF
2764	32	F8000-FFFF
27128	64	F0000-FFFF

With optional iSBC® 341 MULTIMODULE™ EPROM —

Device	Total K Bytes	Hex Address Range
2716	16	FC000-FFFF
2732A	32	F8000-FFFF
2764	64	F0000-FFFF
27128	128	E0000-FFFF

* Four iSBC 88/45 EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs (3 sockets); iSBC 341 sockets also support EPROMs and RAMs.

Timer Input Frequency — 8.00 MHz \pm 0.1%

Interfaces

iSBX™ Bus — All signals TTL compatible

Serial RS232C Signals —

CTS	CLEAR TO SEND
DSR	DATA SET READY
DTE TXC	TRANSMIT CLOCK
DTR	DATA TERMINAL READY
FG	FRAME GROUND
RTS	REQUEST TO SEND
RXC	RECEIVE CLOCK
RXD	RECEIVE DATA
SG	SIGNAL GROUND
TXD	TRANSMIT DATA

Serial RS422A/449 Signals —

CS	CLEAR TO SEND
DM	DATA MODE
RC	RECEIVE COMMON
RD	RECEIVE DATA
RS	REQUEST TO SEND
RT	RECEIVE TIMING
SC	SEND COMMON
SD	SEND DATA
SG	SIGNAL GROUND
TR	TERMINAL READY
TT	TERMINAL TIMING

Electrical Characteristics

DC Power Dissipation — 28.3 Watts

DC Power Requirements —

Configuration	Current Requirements (all voltages \pm 5%)		
	+ 5V	+ 12V	- 12V
without EPROM ¹	5.1A	20 mA	20 mA
with 8K EPROM (using 2716)	+ 0.14A	—	—
with 16K EPROM (using 2732A)	+ 0.20A	—	—
with 32K EPROM (using 2764)	+ 0.24A	—	—
with 64K EPROM (using 27128)	+ 0.24A	—	—

NOTE 1: AS SHIPPED - no EPROMs in sockets, no iSBC 341 module. Configuration includes terminators for two RS422A/449 and one RS232C channels.

Serial Communication Characteristics

Channel	Device	Supported Interface	Max. Baud Rate
A	8274 ¹	RS442A/449	800K SDLC/HDLC
		RS232C	125K Synchronous
		CCITT V.24	50K Asynchronous
B	8274	RS232C	125K Synchronous ²
		CCITT V.24	50K Asynchronous
C	8273 ³	RS442A/449	64K SDLC/HDLC ³
		RS232C	9.6K SELF CLOCKING
		CCITT V.24	

NOTES:

1. 8274 supports HDLC/SDLC/SYNC/ASYNCR multiprotocol
2. Exceed RS232C/CCITT V.24 rating of 20K baud
3. 8273 supports HDLC/SDLC

BAUD RATE EXAMPLES (Hz)

8254 Timer Divide Count N	Synchronous K Baud	Asynchronous		
		$\div 16$	$\div 32$	$\div 64$
10	800	50.0	25.0	12.5
26	300	19.2	9.6	4.8
31	256	16.1	8.06	4.03
52	154	9.6	4.8	2.4
104	76.8	4.8	2.4	1.2
125	64	4.0	2.0	1.0
143	56	3.5	1.7	.87
167	48	3.0	1.5	.75
417	19.2	—	—	—
833	9.6	—	—	—
EQUATION	8,000,000 N	500K N	250K N	125K N

SERIAL INTERFACE CONNECTORS

Interface	Mode ¹	MULTIMODULE™ Edge Connector	Cable	Connector
RS232C	DTE	26-pin ⁴ , 3M-3462-0001	3M ² -3349/25	25-pin ⁶ , 3M-3482-1000
RS232C	DCE	26-pin ⁴ , 3M-3462-0001	3M ² -3349/25	25-pin ⁶ , 3M-3483-1000
RS449	DTE	40-pin ⁵ , 3M-3464-0001	3M ³ -3349/37	37-pin ⁷ , 3M-3502-1000
RS449	DCE	40-pin ⁵ , 3M-3464-0001	3M ³ -3349/37	37-pin ⁷ , 3M-3503-1000

NOTES:

1. DTE — Data Terminal Equipment mode (male connector); DCE — Data Circuit Equipment mode (female connector) requires line swaps.
2. Cable is tapered at one end to fit the 3M-3462 connector.
3. Cable is tapered to fit 3M-3464 connector.
4. Pin 26 of the edge connector is not connected to the flat cable.
5. Pins 38, 39, and 40 of the edge connector are not connected to the flat cable.
6. May be used with the cable housing 3M-3485-1000.
7. Cable housing 3M-3485-4000 may be used with the connector.

Line Drivers (supplied)

Device	Characteristic	Qty	Installed
1488	RS232C	3	1
1489	RS232C	3	1
3486	RS422A	2	2
3487	RS422A	2	2

Reference Manual

143824 — iSBC 88/45 Advanced Data Communications Processor Board Hardware Reference Manual (not supplied).

Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051

ORDERING INFORMATION

Part Number Description

SBC 88/45 8-bit 8088-based Single Board Computer with 3 HDLC/SDLC serial channels

SERIAL INTERFACE CONNECTORS

Connector	Cable	MULTIMODULE™ Edge Connector	Mode	Interface
25-pin, 3M-288-1000	3M-324822	25-pin, 3M-3482-0001	DTE	RS-232C
25-pin, 3M-3483-1000	3M-324822	25-pin, 3M-3482-0001	DCE	RS-232C
25-pin, 3M-3502-1000	3M-324822	25-pin, 3M-3482-0001	DTE	RS-485
25-pin, 3M-3503-1000	3M-324822	25-pin, 3M-3482-0001	DCE	RS-485

NOTES:

1. DTE — Data Terminal Equipment mode (male connector); DCE — Data Circuit Equipment mode (female connector).
2. Cable is labeled at one end to fit the 3M-3482 connector.
3. Cable is labeled to fit the 3M-3483 connector.
4. Cable is labeled to fit the 3M-3502 connector.
5. Cable is labeled to fit the 3M-3503 connector.

October 1979

Reference Manual

14324 — iSBG 8085 Advanced Data Communications Processor Board Hardware Reference Manual (not supplied).

Reference manuals may be ordered from any Intel sales representative, distributor, office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

Line Drivers (supplied)

Device	Characteristic	Qty	Installed
485	RS-232C	2	1
485	RS-232C	2	1
485	RS-232C	2	1
485	RS-232C	2	1

Using the iSBC 544™ Intelligent Communications Controller

Steve Verleye
OEM Microcomputer Systems Applications

I. INTRODUCTION

As the microcomputer system found its way into more and more demanding applications the need became clear for a new and innovative solution to the old problem of providing timely response to real world events. This need was never clearer than in the field of communications where throughput and response time are the keys to success. The iSBC 544 Intelligent Communications Controller (ICC) is the vanguard of a family of intelligent slave computers that provide a unique and powerful answer to the needs of the microcomputer user.

This application note is intended to introduce the reader to the intelligent slave concept in general and the iSBC 544 board in particular. After a brief overview of the evolution of the concept and the features it provides, the hardware and software aspects of the controller are studied. Following this a summary of various system throughput tests is examined to evaluate the performance of the intelligent slave versus more traditional system architectures. We then study two example applications of the product and relate the earlier discussions to the real world. Finally, some system software is presented that handles all data transfer duties between master single board computers and intelligent slaves on the MULTIBUS system bus. More detailed information on many of the topics covered in this note can be found in the related publications listed in the front-piece.

II. OVERVIEW

Intelligent Slave Architecture

Over the years, component technology has increased at a rapid pace going from discrete components (eg. transistors) to integrated circuits (eg. TTL devices) to programmable peripheral controllers (eg. Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter) to fully intelligent slave devices (eg. Intel 8041A Universal Peripheral Interface). At the system level the evolution followed a similar path using the increasing component technology to create more and more powerful system building blocks. The iSBC 508 I/O board used TTL logic to provide digital I/O expansion for iSBC computers. The

iSBC 534 board took advantage of programmable LSI devices to provide a programmable communications expansion board. Now, with the advent of the iSBC 544 Intelligent Communications Controller, a new level of system capability is made possible with the fully intelligent slave controller.

The cornerstone of the intelligent slave architecture is the dual port memory. Through the use of this shared memory space, a fast and efficient protocol can be established to allow for cooperation between master and intelligent slave in solving the needs of the application system. In addition to the shared memory, the CPU on the intelligent slave also has some local RAM and local PROM storage for programs. By using this architecture the advantages of multiprocessing and Direct Memory Access (DMA) controllers are blended together. Unlike DMA controllers, the intelligent slave works totally within its own data space. Therefore, it is not affected by bus traffic nor does it add to this traffic. And, since the on-board CPU gets its instructions from local PROM instead of predefined hard-wired logic or micro-code, the user has total flexibility in defining the functions the intelligent slave will assume in the overall system.

Although the contents of an intelligent slave make it look very similar to a single board computer, the assumption of the slave role provides a distinct advantage. By performing duties for a master single board computer, the slave relieves the master of low-level processing duties and at the same time is itself relieved of system responsibilities.

In order to position the iSBC 544 product and outline what features it brings to the application system it is necessary to define the functions involved with communicating data. The three main functional divisions are illustrated in Figure 1. At the lowest level the physical interconnection is maintained. This level involves such standards as RS232C which defines the requirements for transmitting bits from point to point.

The data transmission level involves the transfer of bytes and/or blocks of data from devices to computers and from node to node in computer networks. The hardware dependent software such as interrupt service and device polling is

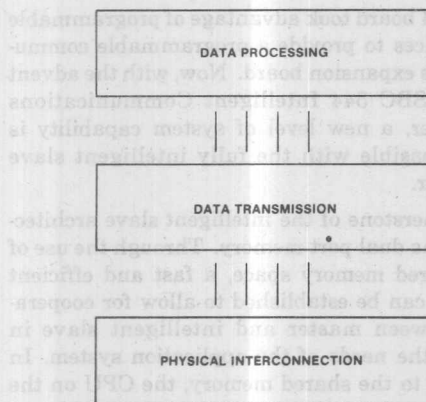


Figure 1. Layering of Communication System Functions

part of this level as are handlers for standard protocols such as SDLC, HDLC, Bisync and X.25 or special purpose schemes and custom protocols.

The highest level performs the actual processing of the data and calls upon the lower levels to move the data from place to place. The application software resides at this level as do some high level software functions such as program to program and process to process communications packages.

Now that we have a map of system functions to guide us, it is possible to gain an understanding of the usefulness of a product like the iSBC 544 Intelligent Communications Controller. If an iSBC 534 board (which contains four USART devices) was included to handle the expansion of serial I/O capacity the mapping of system functions would look like that shown in Figure 2. The four USARTs on the board would handle the physical interconnection but due to the lack of intelligence on the board the master CPU would be burdened with all of the data transmission duties in addition to its real duty, data processing.

When an iSBC 544 board is used in the system, the mapping of system functions is as shown in Figure 3. The physical interconnection is still handled by the USARTs on the board but now the on-board CPU can be programmed to assume the data transmission duties. With an intelligent slave in the system, the master CPU is freed to concentrate on the data processing functions and the end result is that each function in the system is handled in the most efficient manner possible.

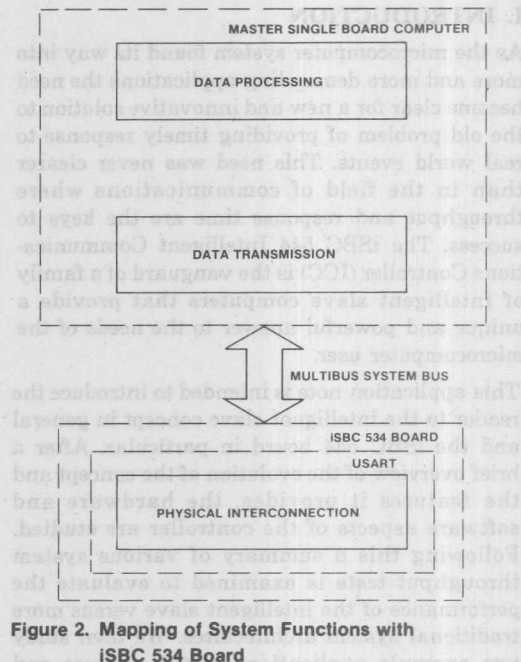


Figure 2. Mapping of System Functions with iSBC 534 Board

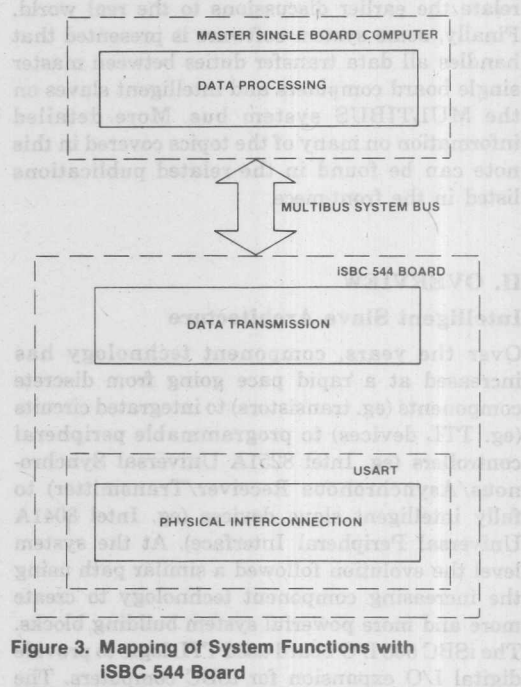


Figure 3. Mapping of System Functions with iSBC 544 Board

The iSBC 544 Board

The iSBC 544 Intelligent Communications Controller contains:

- An Intel 8085A CPU operating at 2.76 MHz.
- Sockets for up to 8K bytes of read only memory (user can choose Intel 2716, 2316E or 2732 devices).
- 16K bytes of dynamic, dual port Random Access Memory (RAM).
- 256 bytes of static local RAM.
- Four Intel 8251A USARTs with programmable baud rates.
- Two Intel 8253 Programmable Interval Timers.
- Intel 8155 parallel interface providing 22 parallel I/O lines and one 14 bit interval timer. Various input and output lines are dedicated to provide an interface to a Bell 801 or equivalent Automatic Call Unit (ACU).
- 8259A Priority Interrupt Controller.

III. HARDWARE CONSIDERATIONS

This section of the application note will focus on the iSBC 544 hardware and will outline the features of the board and its uses. Appendix A contains simplified logic diagrams of the iSBC 544 board which can be referenced in the following discussions.

Two Mode Operation

The iSBC 544 board is capable of operating in one of two modes; 1) intelligent slave and 2) stand-alone communications computer. The mode can either be set with a switch or it can be "toggled" via a software driven flip-flop on the board. In the intelligent slave mode the CPU on the iSBC 544 board operates strictly within its on-board resources. Communications with 8-bit and 16-bit master single board computers is accomplished through the dual port memory. Since the on-board CPU executes code out of its local PROM program storage the system designer is free to define which functions the slave will assume in the system design. As discussed earlier, this could include all or part of the system data transmission duties or could involve application specific duties such as terminal format control, code conversion or terminal input editing.

In the stand-alone mode, the logic on the board disables off board access to the dual port RAM and the bus buffers are used to allow the on-board CPU to access expansion memory and I/O on the MULTIBUS system bus. In this mode the iSBC 544 board drives the bus busy (BUSY/) control line active disallowing any other bus master access to the bus. The stand-alone communications computer is capable of performing all of the functions of the applications system. Referring once again to the diagram of the functions of a communication system, the stand-alone communications computer, with or without system expansion, is responsible for all data transmission and data processing functions. In small applications requiring multiple serial lines the stand-alone iSBC 544 controller is a perfect fit.

In very special circumstances it is possible to share the system bus by toggling the mode set flip-flop between master and slave mode. Figure 4

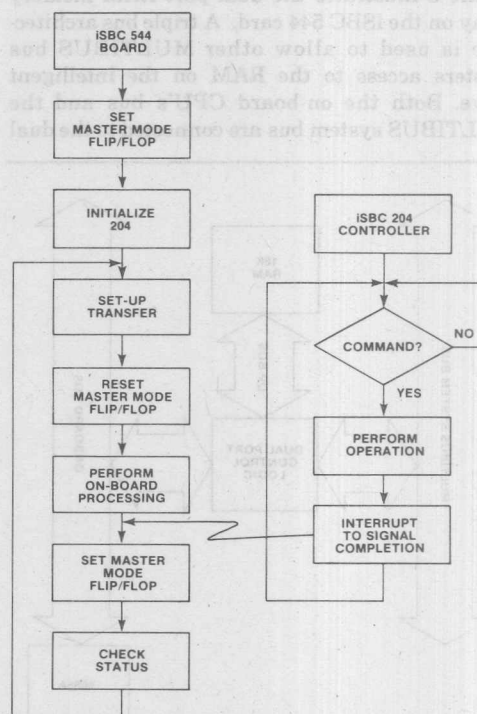


Figure 4. iSBC 544 Controller Running iSBC 204 Disk Controller

shows the flow chart for a routine (code in Appendix B) that makes use of the "software switch" to operate an iSBC 204 Diskette Controller. Using the iSBC 544 board in a system with DMA devices is not recommended except in cases where DMA accesses are short and relatively rare. The use of the CPU for the handling of other system devices could seriously degrade its performance as a communications controller. However, this capability could be extremely useful in a system such as a small message store and forward where the disk traffic is not heavy and including a CPU card just to handle the disk would be wasteful. Use of the "software switch" to share the bus with another iSBC CPU is not advised because of the amount of protocol that would be required to keep the CPUs from interfering with each other on the bus.

Dual Port RAM

Figure 5 illustrates the dual port RAM memory array on the iSBC 544 card. A triple bus architecture is used to allow other MULTIBUS bus masters access to the RAM on the intelligent slave. Both the on-board CPU's bus and the MULTIBUS system bus are connected to the dual

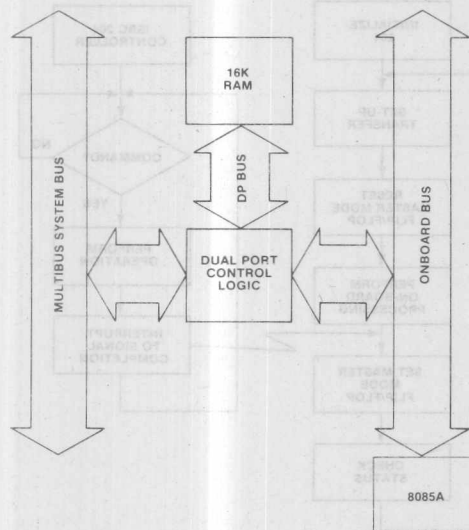


Figure 5. Dual Port Control Logic

port controller. From here the dual port bus is connected to the 16K of dynamic RAM memory. Memory transfer requests from either of the first two busses are handled by the dual port control logic with the on-board CPU being given priority if contention arises. The local CPU is favored so that it is not overly delayed in handling its time critical functions.

The address mapping of the dual port memory on the iSBC 544 is diagrammed in Figure 6. The user can enable access from the MULTIBUS system bus to 0, 4K, 8K or all 16K of the RAM on each iSBC 544 board. The dual port control logic decodes the full 20-bit address and provides an 8-bit data path to the bus. For these reasons the iSBC 544 board is compatible with 8080A, 8085A and 8086 based single board computers. The user can also select the block of addresses on the system bus to which the iSBC 544 RAM will respond.

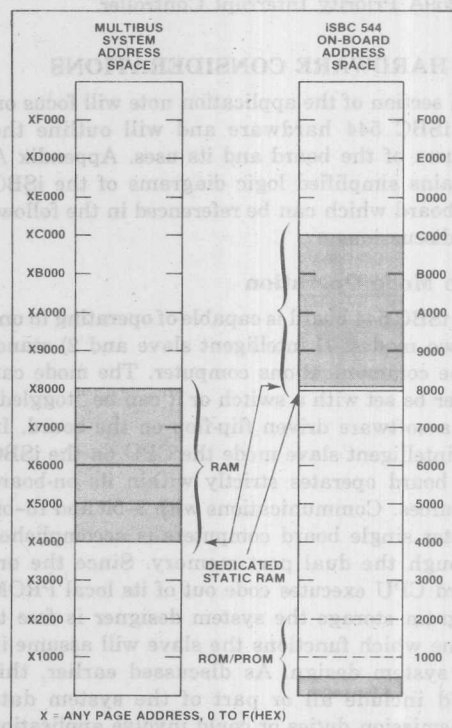


Figure 6. Address Mapping on Dual Port RAM Block

When accessed by the on-board CPU, the dual port RAM always appears at 8000H. If the iSBC 544 board is operating in the stand-alone computer mode, the board is capable of generating the 16-bit bus address supported by the 8085A CPU.

Interrupt Structure

The interrupt structure of the iSBC 544 controller is designed to handle the heavy load imposed by the inherent real-time nature of the communications application. An 8259A Priority Interrupt Controller handles the four receiver and transmitter ready interrupts from the 8251A devices and provides vectored interrupts using one of many available priority schemes. In addition to the eight interrupt sources handled by the 8259 there are various others that can be connected directly to the vectored interrupt inputs on the 8085A (RST 5.5, 6.5, 7.5 and TRAP). One interrupt is generated by the dual port control logic whenever a byte is written into the base address of the dual port memory by an offboard CPU. This interrupt, the flag interrupt, is cleared automatically when the on-board CPU reads the byte and is useful when designing a master-slave protocol since it provides a unique interrupt to each slave in the system.

If the 8251A devices are used to interface to modems the loss of carrier and ring indicator interrupts from all four channels need to be connected to 8085A interrupt request inputs. This is accomplished with four input OR gates tying the eight sources into RST 6.5. The ring indicator and carrier detect lines can also be monitored through a parallel I/O port. This port would be read in a polled system to determine status or could be used along with the OR-tied interrupts to determine which channel is sourcing the current interrupt.

The remaining interrupt sources come from the extra timer/counters and from the MULTIBUS interrupt lines. In addition to receiving interrupts from the bus, the iSBC 544 board has the capability of generating MULTIBUS interrupts using the Serial Output Data (SOD) line on the 8085A CPU.

Modem and Autocall Interface

The iSBC 544 controller uses 8251A and 8155 devices for interface to modems and an autocall

unit respectively. All of the necessary handshaking signals concerned with the modem interface are connected to the 8251A and the carrier detect and ring indicator signals, as previously mentioned, can be connected to interrupt inputs. The 8155 parallel ports are wired as shown in Figure 7. All of the commonly used signals defined in the EIA RS-366 specification for interface to an autocall unit are provided. The software necessary for handling the ACU becomes a simple matter of responding to the ACU requests and sending out the BCD digits representing the number being dialed. In addition to the ACU interface, the 8155 monitors various signal states and provides software reset capabilities for the USARTs and some interrupts.

IV. SOFTWARE CONSIDERATIONS

Software for the iSBC 544 ICC falls into three main categories; device programming, master-slave protocols, and communications support. Each of these three topics is covered in the following section with the aim of defining the software requirements and functions of the iSBC 544 board.

Device Programming

The main sources of the power and flexibility of this product are the programmable LSI devices on the board. The first duty of the on-board software is programming these devices to handle the specific task at hand. To start with, the 8251A USART can be programmed for synchronous or asynchronous operation. In synchronous mode the user specifies even, odd or no parity and either external or internal sync detect with one or two sync characters. In the asynchronous mode the programmer selects the parity, the character length (5, 6, 7 or 8 data bits), the framing control (1, 1½ or 2 stop bits) and the baud rate scaling factor (input clock frequency divided by 1, 16 or 64).

The 8253 Programmable Interval Timers provide the receiver and transmitter clocks for the USARTs and, along with the 8251A baud rate scaling factor, are programmed by the software to provide the desired communications frequency. In addition, two additional 16 bit timers are left available to the applications programs to be used as event counters, real-time interrupts, etc.

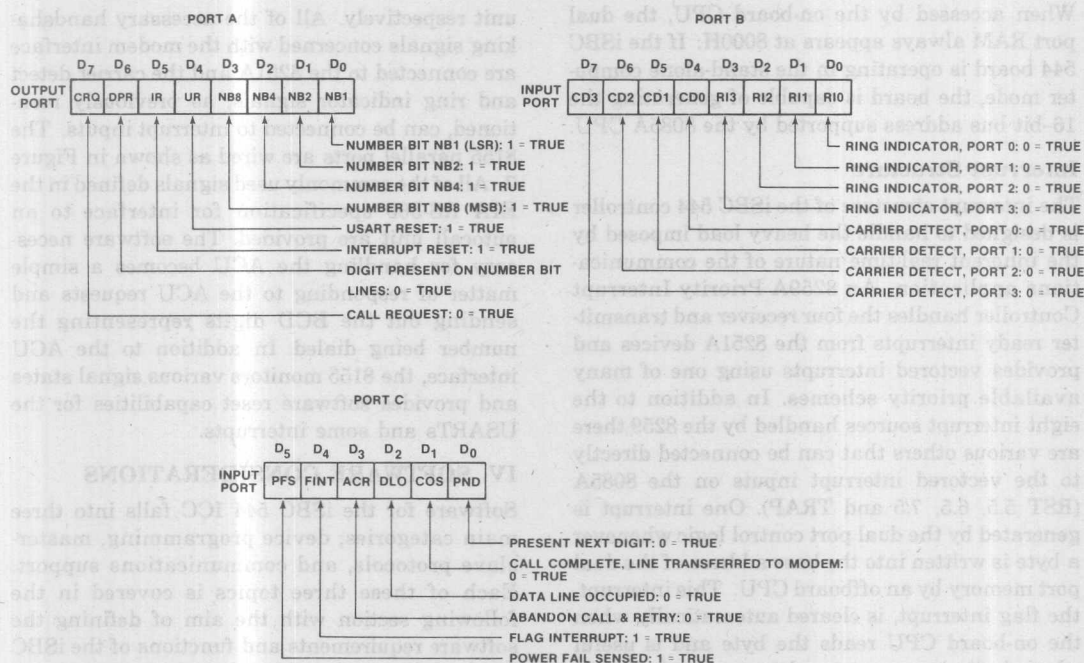


Figure 7. 8155 Pinout Definitions

The 8259A Priority Interrupt Controller is programmed to vector all interrupts through a jump table in memory. Also, the device provides software selectable priority schemes and an interrupt mask register for sophisticated interrupt management designs.

Last, but not least, the 8155 Programmable Peripheral Interface provides various software controlled input and output ports as discussed in previous sections. One specific point to remember is that the power on state of the 8155 clamps the reset signal to the USARTs active and must be removed by programming the 8155 before communications can begin.

Master-Slave Protocols

If an application system is visualized at the highest level it appears to be a computer with various inputs and outputs as depicted in Figure 8a. If this computer is broken down into a master CPU and one or more intelligent slaves, great increases in efficiency and system throughput

can be realized by distributing the duties between the CPUs (Figure 8b). Once this split is performed, some well defined means of communication between master and slaves needs to be defined so that the processes that execute on the different machines can cooperate. This means of communication takes the form of a protocol followed by both master and slave.

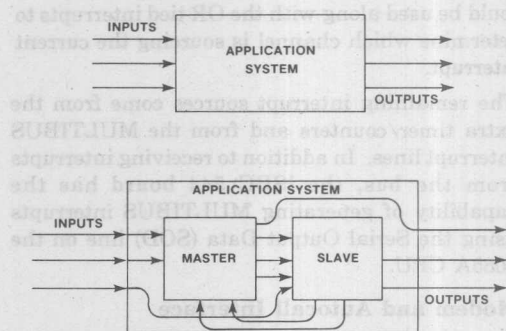


Figure 8a and 8b. System Software Block Diagrams

The intelligent slave architecture was designed to simplify the development of the necessary protocol. The shared memory space in the dual port RAM provides a large communications buffer area where data and commands can be transferred using normal memory transfers. Data structures of any needed complexity can be built in this memory area and accessed by both master and slave. The flag interrupt can be used to provide a unique synchronization signal from a master to a given slave. In addition, the MULTI-BUS interrupt lines can be used to provide extra signals in both directions. As we shall see in the system software section, these basic tools can be utilized to design a general purpose data transfer mechanism which isolates the applications processes from the worries of protocols and synchronization.

Communications Support

The previous software topics dealt mainly with the system overhead that must be handled by the communications processor. The larger and more important duty of the CPU is dealing with the application at hand—communications.

When configured as an intelligent slave to some master iSBC CPU board, the iSBC 544 board works to offload the master of communications related functions and at the same time is itself relieved of a major share of the system overhead and can be tuned to provide the highest possible throughput. With this combination, more complex applications can be tackled where the number of lines and the line frequencies are greatly increased. Multiple systems can be employed to provide a network facility with the iSBC 544 board now handling the network protocol in addition to its other duties. The architecture of the iSBC 544 controller is designed to simplify the user's software development process. The board can be programmed to handle many possible data transmission functions from simple line protocols to terminal control to link protocols and all the way up to network protocols.

In the stand-alone mode, the iSBC 544 board can assume total responsibility for the application. This can be done with on-board resources only or can include the support of offboard expansion like the iSBC 534 four channel serial controller. Appli-

cations of the stand-alone controller could include cluster controllers, peripherals managers, line concentrators or any other small system.

V. THROUGHPUT ANALYSIS

This section of the application note deals with studies that have been done to quantify the performance of the iSBC 544 board in both the stand-alone and intelligent slave modes. After describing the various test configurations and assumptions the data will be presented in graphical form and analyzed. The graphical data can be found in Appendix C.

Stand Alone Throughput

The first two tests were run to determine the absolute best case throughput of the iSBC 544 board configured as a stand-alone computer. Figure 9a shows the iSBC 544 controller continuously outputting data from four buffers to the four USARTs. Figure 9b shows essentially the same setup with eight channels, four on the iSBC 544 board and four on the iSBC 534 expansion card. In each configuration the 8251A was run in synchronous mode and the baud rate was incremented until the transmitter empty signal from the USARTs became active. Further increments of the baud rates would not have resulted in higher throughput since the CPU was already spending 100% of its available time responding to USART service requests.

The maximum rate for the first configuration (iSBC 544 board only) was 32,311 baud per channel. When the iSBC 534 expansion board was added a rate of 12,186 baud per channel was achieved. The drop in baud rate was due to the extra processing required by the offboard logic (eg. reading 8259 interrupt controller on the iSBC 534 board to determine which device is requesting service).

It should be noted that the serial throughput tests were run with almost no overhead and no actual processing of the data involved. The reader is expected to apply information on the amount of overhead expected in each individual application. For instance, if the application code for a given system is expected to utilize approximately 40% of the available CPU time and we wish to run four

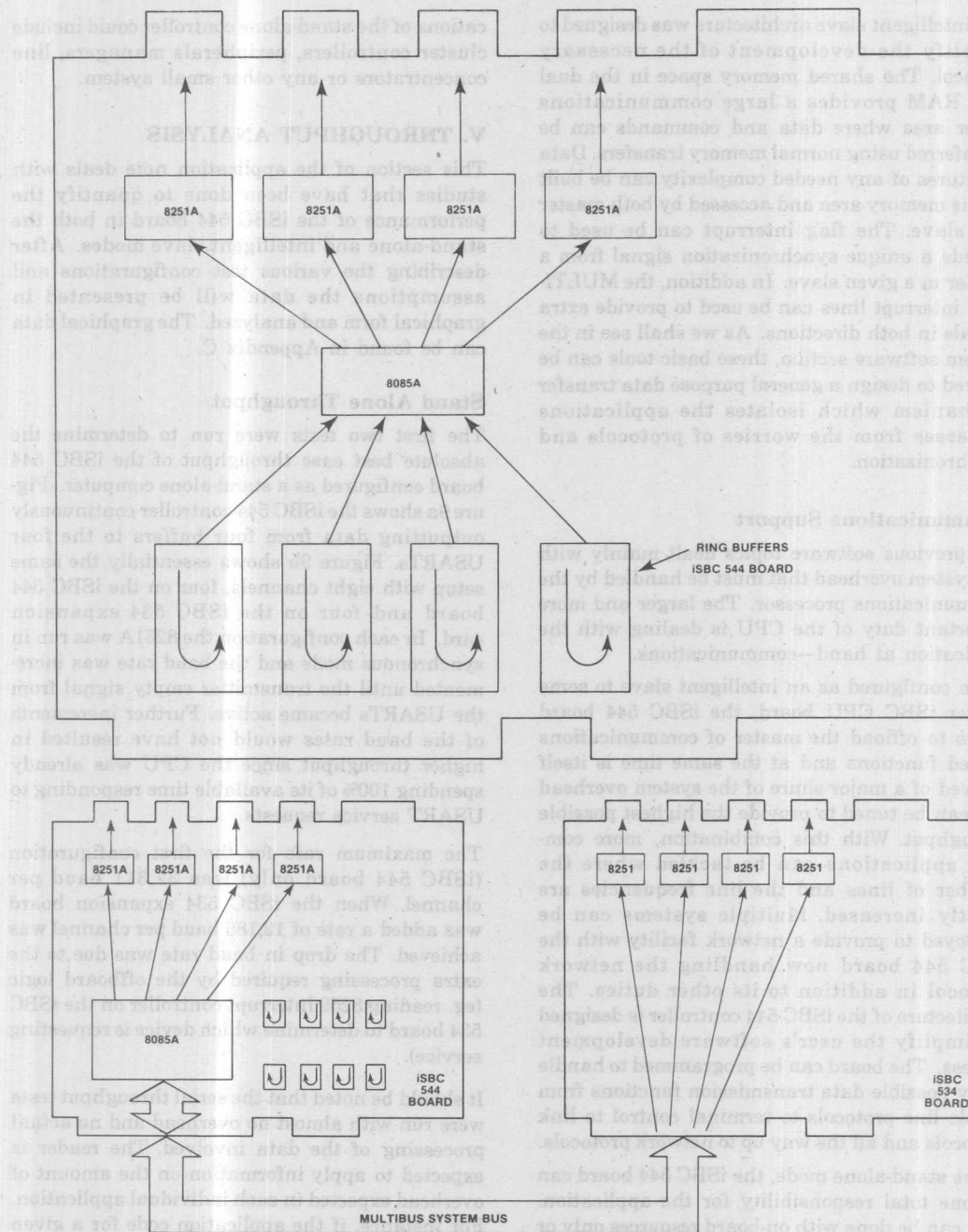


Figure 9a and 9b. Stand-Alone Throughput Configurations

full duplex channels in asynchronous mode the estimate of maximum baud rate would take the following form.

$$\begin{aligned} 32,331 \text{ baud per channel} - 40\% &= 19,398.6 \text{ baud} \\ 19,398.6 \text{ baud per channel synchronous} \times 10/8 &= 24,248.25 \text{ baud asynchronous} \\ 24,248.25 \text{ baud per channel half duplex}/2 &= 12,124.125 \text{ full duplex} \end{aligned}$$

Therefore, the maximum standard baud rate would be 9600 baud per channel in full duplex asynchronous mode.

Intelligent Slave Throughput

The remaining four configurations were set up to determine the effectiveness of the intelligent slave in the overall system. The general system configuration is illustrated in Figure 10. The boards surrounded by the box represent the systems under test. The disk controller and two iSBC 80/20 single board computers were active on the bus to simulate the normal bus traffic load in an application system. Various bus duty cycles were created using the computers and the disk controller to perform tasks that resulted in fixed bus utilization.

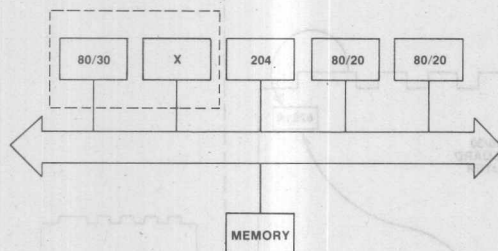


Figure 10. General System Configuration for Throughput Testing

In each configuration a single full duplex channel was set up with the input provided by another CPU. Only those functions dealing with system overhead were included and the data measured

reflected the amount of bus time, master CPU time and slave CPU time left available to applications oriented tasks. In each case this percentage of time available was measured as the baud rate was stepped up so that a graph could be constructed showing time available as a function of transmission speed.

CPU free time was measured using a counting program running in the background. After each USART interrupt the counter was started. As interrupts from other sources came in the counting was preempted and then resumed after servicing the interrupt. When the next USART interrupt occurred, the counter contents were examined and if the value was lower than the stored value the current value became the stored value. After ten minutes the stored value was retrieved and used as an indicator of the worst case time available between interrupts.

System bus utilization was measured using the circuit shown in Figure 11. The voltage measured by the digital voltmeter represented a time average of the voltage at the output of the flip-flop. A calibration chart was created using a pulse generator to simulate various duty cycles and then this chart was used to measure bus activity while the test was running.

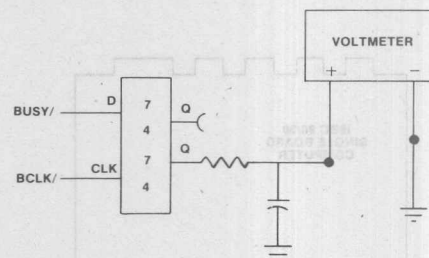


Figure 11. Bus Free Time Measurement Circuit

Configuration 1 is shown in Figure 12. This system uses a typical method of communications expansion with the iSBC 80/30 single board computer handling the lines directly via the serial I/O ports on the iSBC 534 I/O controller board.

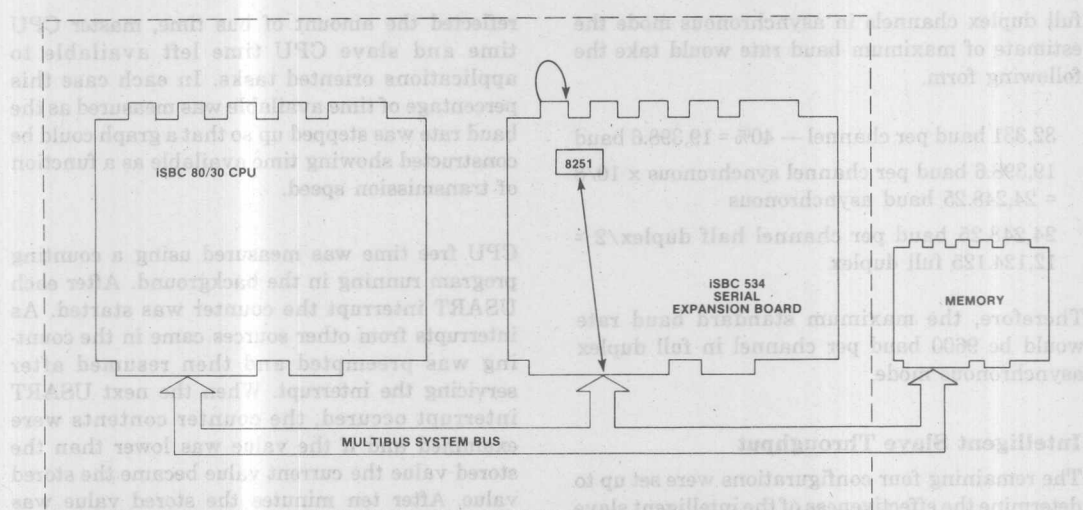


Figure 12. System Throughput Test, Configuration 1

The second configuration (Figure 13) illustrates the performance of the traditional DMA controller approach. If the communications controller had DMA logic instead of a dual port memory and transferred data directly into system memory the performance would be as observed in this test.

In configuration 3 (Figure 14) the iSBC 544 board was used in the intelligent slave mode. This

configuration differs from the second in that memory transfers involved only local memory and bus access was not required on a per character basis.

The fourth and final configuration sought to identify the loading that additional intelligent slave controllers would impose on master CPU time and bus free time. Figure 15 shows the

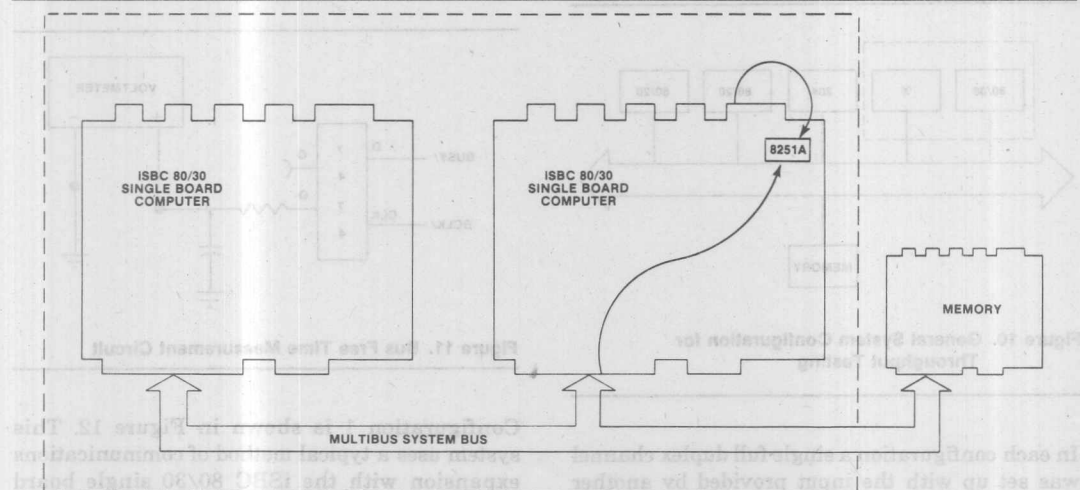


Figure 13. System Throughput Test, Configuration 2

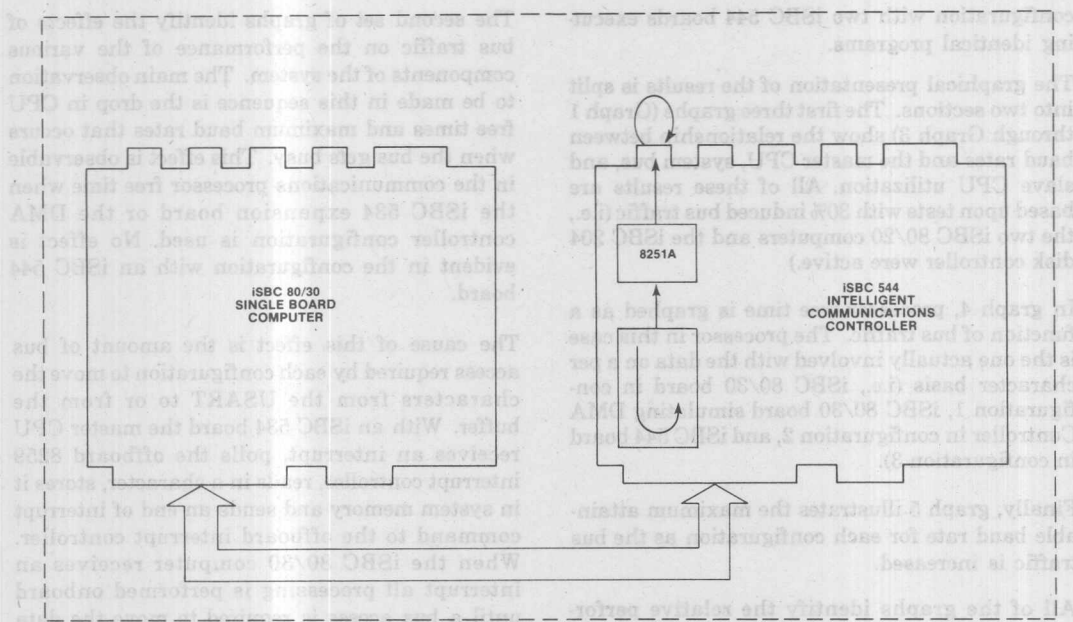


Figure 14. System Throughput Test, Configuration 3

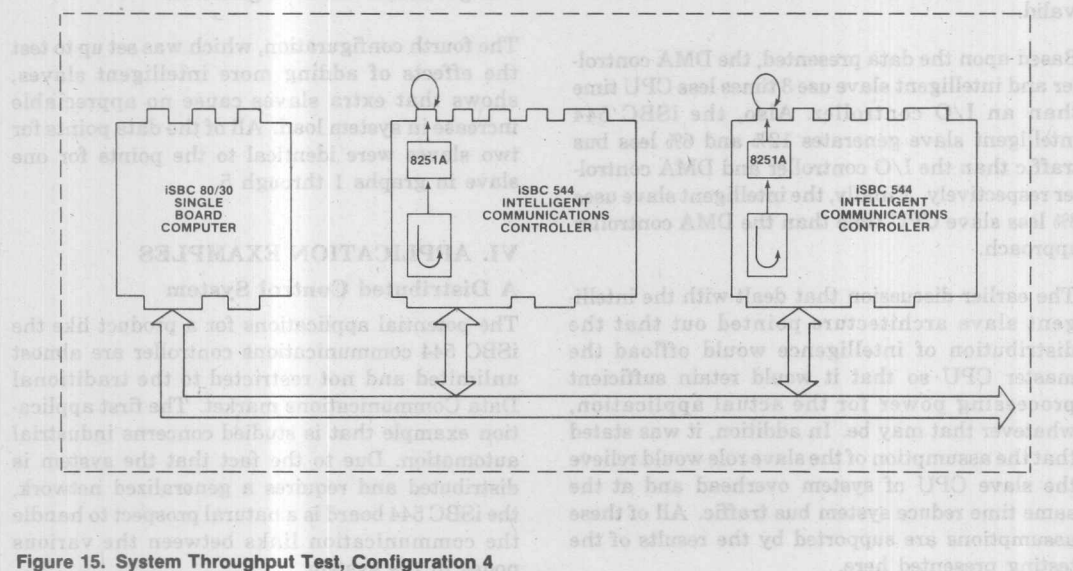


Figure 15. System Throughput Test, Configuration 4

configuration with two iSBC 544 boards executing identical programs.

The graphical presentation of the results is split into two sections. The first three graphs (Graph 1 through Graph 3) show the relationship between baud rates and the master CPU, system bus, and slave CPU utilization. All of these results are based upon tests with 30% induced bus traffic (i.e., the two iSBC 80/20 computers and the iSBC 204 disk controller were active.)

In graph 4, processor free time is graphed as a function of bus traffic. The processor in this case is the one actually involved with the data on a per character basis (i.e., iSBC 80/30 board in configuration 1, iSBC 80/30 board simulating DMA Controller in configuration 2, and iSBC 544 board in configuration 3).

Finally, graph 5 illustrates the maximum attainable baud rate for each configuration as the bus traffic is increased.

All of the graphs identify the relative performance difference between the configurations. Absolute numbers are not presented due to the fact that the overhead imposed by the test software affects the CPU time being measured. Since the overhead applies equally to all configurations, the relative performance indications are valid.

Based upon the data presented, the DMA controller and intelligent slave use 3 times less CPU time than an I/O controller. Also, the iSBC 544 intelligent slave generates 12% and 6% less bus traffic than the I/O controller and DMA controller respectively. Finally, the intelligent slave uses 8% less slave CPU time than the DMA controller approach.

The earlier discussion that dealt with the intelligent slave architecture pointed out that the distribution of intelligence would offload the master CPU so that it would retain sufficient processing power for the actual application, whatever that may be. In addition, it was stated that the assumption of the slave role would relieve the slave CPU of system overhead and at the same time reduce system bus traffic. All of these assumptions are supported by the results of the testing presented here.

The second set of graphs identify the effects of bus traffic on the performance of the various components of the system. The main observation to be made in this sequence is the drop in CPU free times and maximum baud rates that occurs when the bus gets busy. This effect is observable in the communications processor free time when the iSBC 534 expansion board or the DMA controller configuration is used. No effect is evident in the configuration with an iSBC 544 board.

The cause of this effect is the amount of bus access required by each configuration to move the characters from the USART to or from the buffer. With an iSBC 534 board the master CPU receives an interrupt, polls the offboard 8259 interrupt controller, reads in a character, stores it in system memory and sends an end of interrupt command to the offboard interrupt controller. When the iSBC 80/30 computer receives an interrupt all processing is performed onboard until a bus access is required to move the data byte from/to memory. In the case of the intelligent slave, all processing for a character is performed onboard. Thus, as the system bus becomes very fully utilized, the delays encountered in receiving bus access by the first two configurations become significant.

The fourth configuration, which was set up to test the effects of adding more intelligent slaves, shows that extra slaves cause no appreciable increase in system load. All of the data points for two slaves were identical to the points for one slave in graphs 1 through 5.

VI. APPLICATION EXAMPLES

A Distributed Control System

The potential applications for a product like the iSBC 544 communications controller are almost unlimited and not restricted to the traditional Data Communications market. The first application example that is studied concerns industrial automation. Due to the fact that the system is distributed and requires a generalized network, the iSBC 544 board is a natural prospect to handle the communication links between the various nodes in the system.

Design Requirements

The system to be designed is intended to provide the framework for a family of distributed control systems where the configurations and the objects to be controlled vary from system to system. Figure 16 shows the general picture of the system.

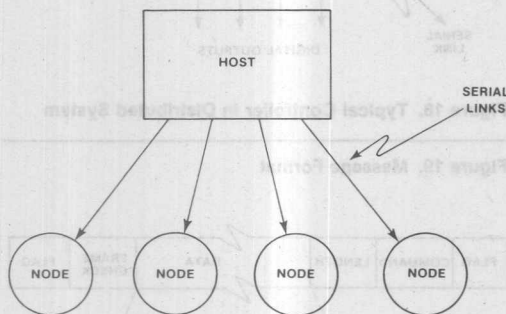


Figure 16. General Diagram of Distributed Control System

The host is responsible for providing supervisory control and a high-level human interface. The system can be expanded as shown in Figure 17 where the controllers attached to the host are replaced by intermediate nodes which contain controllers or other nodes. This process can be continued as far as is necessary to provide the needed number of controllers. Each controller in the diagram represents a localized closed loop control system that is tailored to the specific application.

The following system requirements need to be met by the computer network:

- The host CPU must have sufficient computational power to handle the human interface, mass storage management, supervisory control calculations and network control.
- The host CPU must not be overly burdened by low-level communications functions if it is to handle the other duties assigned to it.
- Node controllers must be capable of handling 8 medium speed lines and also modems and autocal units since the nodes or controllers attached may be remote.

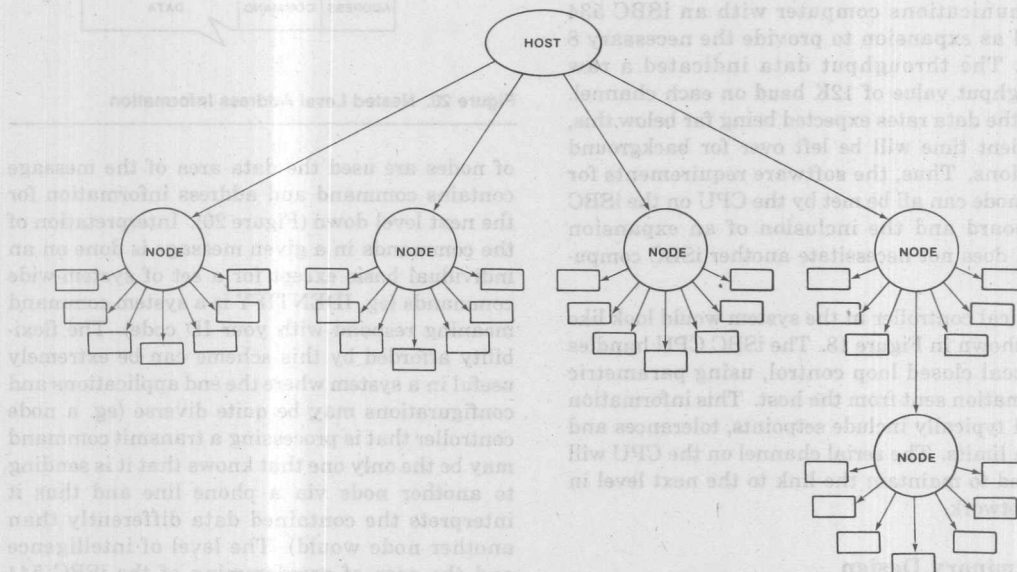


Figure 17. Expanded Diagram of Distributed Control System

- The message transmission format must be independent of the configuration and end application. The nodes in the network must be capable of passing through messages with and without interpreting the contained data.
- The system must be capable of auto-configuration (since the network configuration is tailored to the specific application, the host must be able to automatically determine the setup at power on).
- Each node controller is responsible for verifying the integrity of the nodes attached.

System Configuration

Based upon the design criteria and the benchmark information the chosen configuration uses an iSBC 86/12 Single Board Computer as the host with an iSBC 544 intelligent slave handling the communications load for the CPU. The USART on the CPU board will talk to the local terminal and an iSBC 206 Hard Disk Controller will be used to provide up to 40 Megabytes of mass storage capacity.

The requirements for the node controllers point to an iSBC 544 board configured as a stand-alone communications computer with an iSBC 534 board as expansion to provide the necessary 8 lines. The throughput data indicated a raw throughput value of 12K baud on each channel. With the data rates expected being far below this, sufficient time will be left over for background functions. Thus, the software requirements for each node can all be met by the CPU on the iSBC 544 board and the inclusion of an expansion board does not necessitate another iSBC computer.

A typical controller in the system would look like that shown in Figure 18. The iSBC CPU handles the local closed loop control, using parametric information sent from the host. This information would typically include setpoints, tolerances and alarm limits. The serial channel on the CPU will be used to maintain the link to the next level in the network.

Preliminary Design

The message format that the system uses is shown in Figure 19. When multiple nested levels

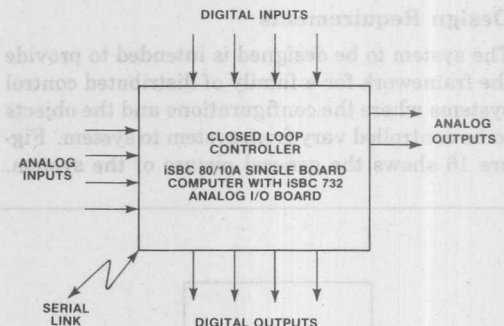


Figure 18. Typical Controller in Distributed System

Figure 19. Message Format

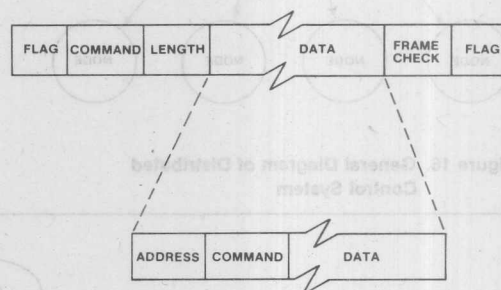


Figure 20. Nested Level Address Information

of nodes are used the data area of the message contains command and address information for the next level down (Figure 20). Interpretation of the commands in a given message is done on an individual basis except for a set of system-wide commands (eg. IDENTIFY is a system command meaning respond with your ID code). The flexibility afforded by this scheme can be extremely useful in a system where the end applications and configurations may be quite diverse (eg. a node controller that is processing a transmit command may be the only one that knows that it is sending to another node via a phone line and thus it interprets the contained data differently than another node would). The level of intelligence and the ease of programming of the iSBC 544 board make this generalized transmission scheme possible.

The simplest means of auto-configuration requires each controller in the system to send an identity message to the nearest node. This node would know the logical address of the controller that sent the message and would attach this address to the message and retransmit it to the next level as illustrated in Figure 21. This process would be repeated until the host is reached and would contain, at this point, all necessary address information to reach the given controller.

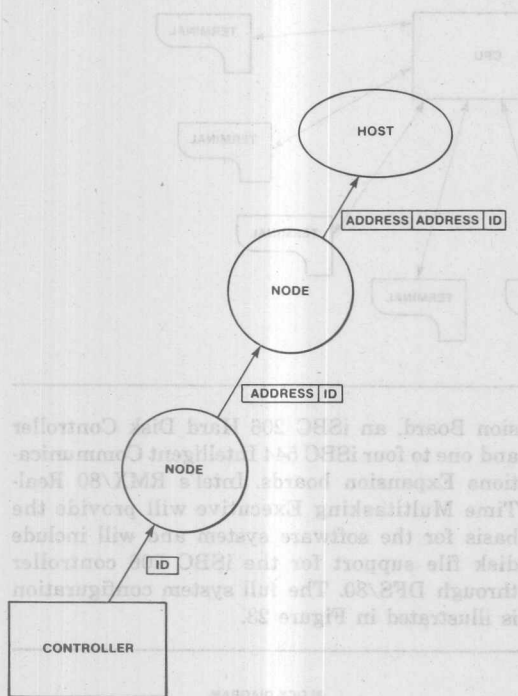


Figure 21. Auto Configuration

The human interface on the host would provide a mapping mechanism to attach meaningful symbolic names to the various nodes in the system. This labeling, along with the application specific control algorithms, make it possible to say something like "lower the temperature on the third floor to 68°F". The host breaks this information down into setpoints and tolerances,

uses the map to determine the path to the node(s) responsible for the third floor and transmits the information through the network.

Each node controller in the system has the added responsibility of verifying the integrity of all the nodes attached to it. This duty can be handled by periodic background commands issued from the host and propagated through the network. Each node is responsible for passing the command along and also polling the nodes attached to it and reporting back any error conditions.

Summary

Through the use of a powerful 16-bit iSBC Single Board Computer, various low-cost 8-bit iSBC CPUs and the iSBC 544 communications controller, a flexible and extensible distributed control system is easy to design. The dual nature of the iSBC 544 board provides both an intelligent front end to the host computer and a high-speed stand-alone nodal concentrator. The ability to individually customize the software on each controller provides for an easily expandable system design.

Terminal Cluster Controller

The second application example concerns itself with a terminal cluster controller. The system shown in Figure 22 uses a number of "dumb" terminals and makes them appear "intelligent" via a local microcomputer system. The local microcomputer interfaces with the operator and accesses a local data base to provide an inquiry and data entry service. When necessary, the local microcomputer is capable of calling the host via an autocal unit and exchanging information and updates to the data base.

Design Criteria

The terminal cluster controller must meet the following criteria:

- Support must be provided for from four to sixteen operator terminals all running at rates up to 2400 baud.
- Line editing on input must be provided (delete characters, delete lines and pause output).

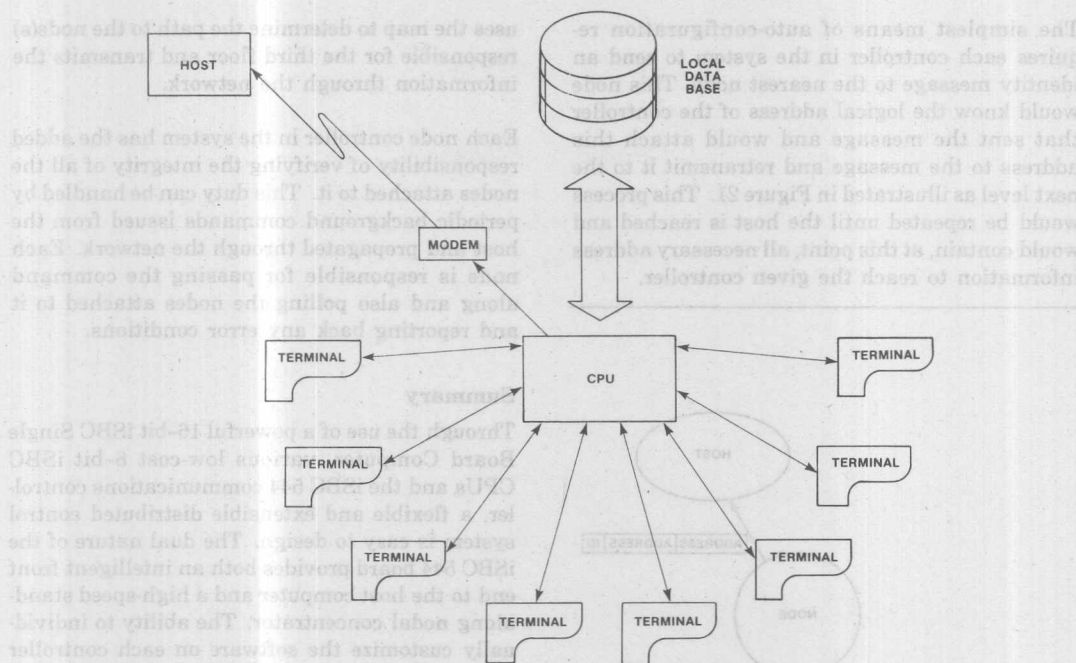


Figure 22. Terminal Cluster

- Support for the terminals must be configurable in that certain stations may require different screen formats.
- Support for an optional hard copy device must be allowed for.
- A considerable amount of CPU free time must be available after the basic terminal facilities are included. This is due to the fact that the data base management software to be written to run on the master single board computer will be extensive.
- Type ahead would be a desired feature since the processing on the master CPU after a line of input has been transmitted may cause a delay in responding and we would like to have the ability to continue entering input while waiting for the response.

System Configuration

The specific iSBC products needed to implement the system described are the iSBC 80/30 Single Board Computer with an iSBC 032 RAM Expan-

sion Board, an iSBC 206 Hard Disk Controller and one to four iSBC 544 Intelligent Communications Expansion boards. Intel's RMX/80 Real-Time Multitasking Executive will provide the basis for the software system and will include disk file support for the iSBC 206 controller through DFS/80. The full system configuration is illustrated in Figure 23.

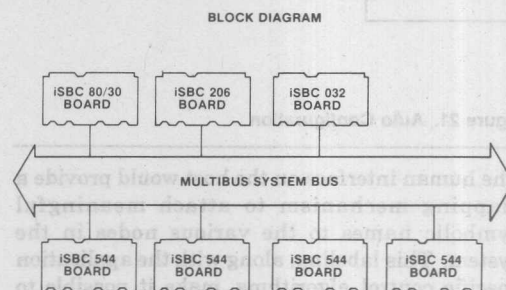


Figure 23. Terminal Cluster Controller System Configuration

Preliminary Design

The first design decision to be made involves the distribution of system functions. Due to the requirements for line-editing and type-ahead the software for processing characters input from the terminal keyboards will be somewhat lengthy. The standard terminal output handler will be very small but provisions for special screen format controls and/or hard copy devices must be allowed for. All of these requirements lead to the use of the iSBC 544 controller for all terminal functions. If the master CPU were burdened with all of these duties it would be unable to adequately perform its data base management functions. The fast CPU and 8K PROM capacity of the iSBC 544 board will be more than adequate for the task at hand.

The throughput tests indicate that the loading imposed by expanding the number of terminals (and therefore the number of iSBC 544 boards) will not adversely affect the performance of the rest of the system. Master CPU free time and bus traffic data for two intelligent slaves in the system were identical to the numbers for one slave. Thus, since the iSBC 80/30 single board computer and the MULTIBUS system bus can handle one iSBC 544 controller they can also handle the maximum of four controllers that may be required by this application. The only observable effect will be caused by the load the extra operators impose on the data base software itself.

The software needed for the iSBC 544 board is now defined and divided into three major pieces; a terminal input handler, a terminal output handler and system software to support the handlers. Since the input and output handlers are invoked via USART interrupts, all that need be done is to write a single routine for each handler and have it talk to all of the devices on the board. This can be accomplished by vectoring the proper interrupts to the entry point of the routine and then polling the 8259A interrupt controller to determine which device needs servicing.

The standard terminal input handler needs to read in the available character from the USART,

check it to see if it is a special command character and, if not, store it into a buffer. If a command character is encountered, the handler will respond by performing the appropriate operation.

The standard terminal output handler simply takes characters out of a buffer upon interrupt from the transmitter and sends them to the appropriate USART. If a different output handler needs to be substituted for a special terminal or a hard copy device, a new routine can be included by modifying the interrupt vector address in the 8259A jump table.

Since the RMX/80 Real-Time Multitasking Executive is being utilized on the master CPU it is desirable to create an RMX/80 handler for the iSBC 544 boards that accepts and processes normal terminal handler request messages. In this manner, application tasks that formerly communicated with the on-board USART via the RMX/80 Terminal Handler can be made to talk to one of the devices on the iSBC 544 board by simply changing the address of an exchange. The following paragraphs, as well as paragraphs in the section on system software, assume a knowledge of the RMX/80 Real-Time Executive. This knowledge is not necessary to use the information contained in this application note. Interested readers are referred to the RMX/80 references listed in the front-piece.

Since this application can have from one to four iSBC 544 boards the RMX/80 driver will need to be configurable. A set of tasks and exchanges will be created for each terminal in the system. One task and exchange pair will accept and process terminal input request messages while another pair will process terminal output requests.

The remaining piece of software that is needed by this system will provide the means for getting commands and data between the master and intelligent slave. Since this is a common need in any system utilizing an intelligent slave we will develop a general purpose scheme that can be used by any application. In this manner, a routine such as the terminal input handler can be written without any concern for how it will get the data it is inputting to the master CPU; all it need do is call upon a standard routine to "transmit"

the data. With these thoughts in mind, the following section discusses the system software developed for master-intelligent slave communication. After the discussion of the system software we will revisit the software for the second application as an example of the use of the data transfer routines.

VII. SYSTEM SOFTWARE

In the earlier discussion of master-slave protocols, the notion was presented of developing a general purpose data transfer scheme which would enable the applications routines on both the slave and master to operate without concerning themselves with protocols and synchronization. This scheme can be implemented by designing a set of primitive routines to handle the data transfer activities. Thus, Figure 8b is expanded as shown in Figure 24 and the applications processes now call upon the primitives to handle the communications between the master and the slave.

Data Transfer Primitives

The basic mechanism used by this implementation of the primitives is a wraparound queue as shown in Figure 25. Each 8251A device has associated with it, in dual port memory, an input and an output queue each of which have a *give*

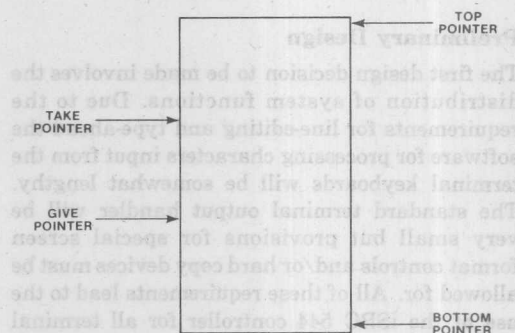


Figure 25. Wrap-around Queue Used by Data Transfer Primitives

and a *take* pointer. The *give* pointer contains the address of the next location in the queue that is available for filling with data. The *take* pointer contains the address of the next byte in an output queue that has been filled and is available. A queue is empty when the *give* and *take* pointers are equal and it is full when the act of incrementing the *give* pointer would make it equal to the *take* pointer. A *wrap* function is defined to increment a pointer such that an increment past the bottom of the queue "wraps" the pointer around to the top of the queue.

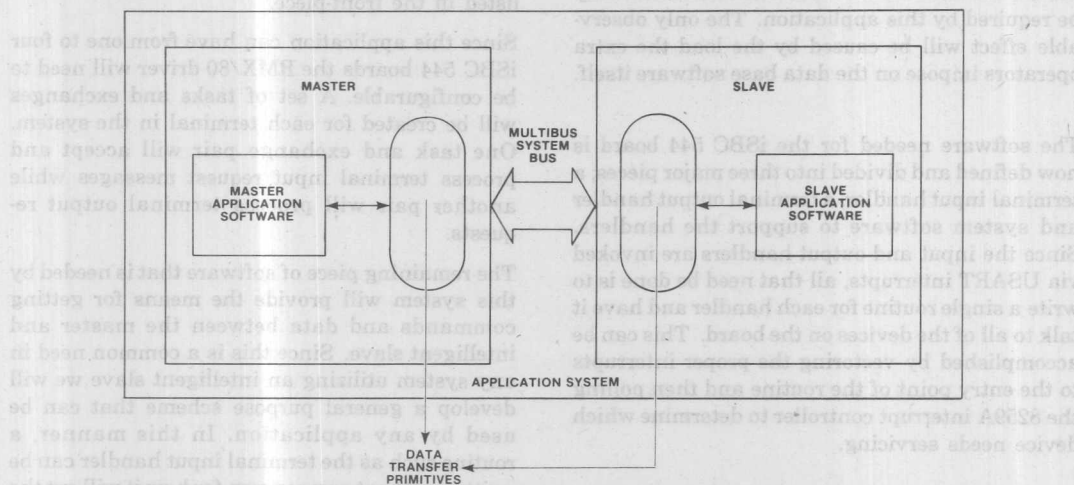


Figure 24. System Software Diagram with Data Transfer Primitives

The primitives all make use of a queue information block located at the base address of the slave's dual port memory (Figure 26). All pointer information is base relative to accommodate the needs of the two CPUs who have different memory maps. The two flag bytes carry information for master-slave and slave-master synchronization signals.

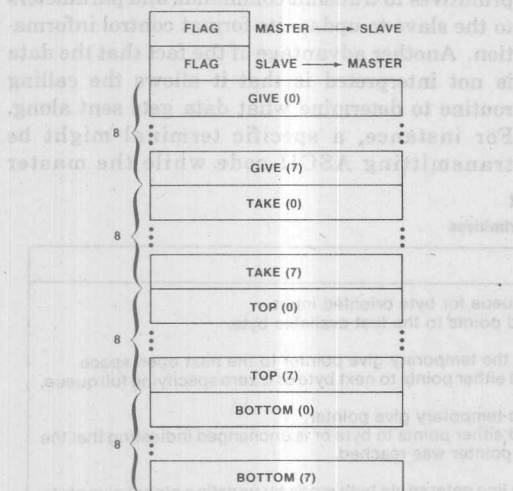


Figure 26. Queue Information Block

The set of primitives provides two distinct methods of information transfer, line oriented and byte oriented. The line oriented primitives are listed in Table 1. Both *get\$line* and *send\$line* transfer information between the queues and buffers provided by the caller. The disadvantage of this scheme is the number of memory moves needed to transfer information. The advantages of the line oriented method are the relative efficiencies and the simplicity of the interface from the calling routine.

The byte oriented primitives (Table 2) allow the calling routine to transfer data directly into and out of the queues. An example of the sequence for putting a character into a queue is illustrated in Figure 27. The routine servicing the receiver ready interrupt calls *next\$space* to get a pointer to the next available slot in the queue and then uses this pointer to transfer the data byte directly into the queue. The *new\$line*, *xmit*, *open\$line* and *receive* primitives are necessary since the global *give* and *take* pointers cannot be modified until all manipulations on the affected section of the queue are complete. If the pointers were modified continuously the routine gathering the data from the other side may see invalid data.

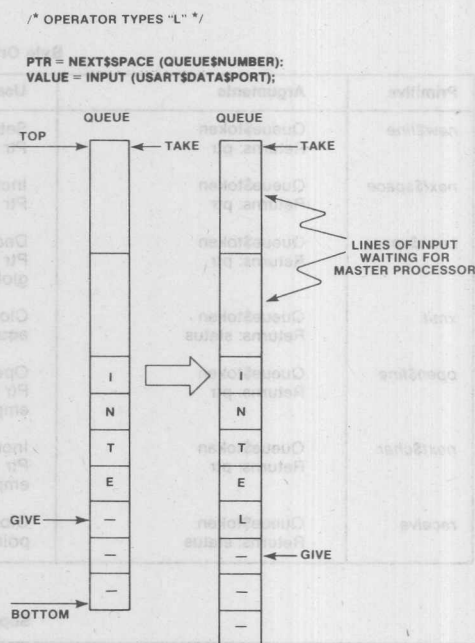


Figure 27. Sequence for Putting Data Into Queue

Table 1

Line Oriented Primitives

Primitive	Arguments	Usage
<i>send\$line</i>	Queue\$token, buf\$ptr, count Returns: overflow	Inserts count characters into queue from buffer If insufficient room available, overflow indicates how many would not fit
<i>get\$line</i>	Queue\$token, buf\$ptr, count Returns: Actual	Retrieves count characters from queue and puts them in buffer Actual indicates how many were actually moved

The remaining primitive routines deal with the general purpose needs of the application software with regard to interrupts, initialization and status checking. A full list of these support routines is contained in Table 3.

There are many features of this implementation and a few of them should be pointed out at this time. By defining a general purpose set of primitive routines to handle the data transfer, the actual means by which the bytes are transferred between slave and master is not visible to the calling routine. If the actual mechanism used needs to be altered the change will not affect the application software as long as the same external interface is maintained.

Another important feature of the primitive routines is the fact that they do not interpret the bytes that are sent to them. Due to this fact, the applications routines are free to send commands and parameters interspersed with the actual data. As an example, the terminal driver on an iSBC 544 board might perform format control based upon table information. The master applications software could use the data transfer primitives to transmit commands and parameters to the slave to update its format control information. Another advantage of the fact that the data is not interpreted is that it allows the calling routine to determine what data gets sent along. For instance, a specific terminal might be transmitting ASCII code while the master

Table 2
Byte Oriented Primitives

Primitive	Arguments	Usage
<i>new\$line</i>	Queue\$token Returns: ptr	Sets up a queue for byte oriented input. Ptr returned points to the first available byte.
<i>next\$space</i>	Queue\$token Returns: ptr	Increments the temporary give pointer to the next open space. Ptr returned either points to next byte or is zero specifying full queue.
<i>back\$space</i>	Queue\$token Returns: ptr	Decrements temporary give pointer. Ptr returned either points to byte or is unchanged indicating that the global give pointer was reached.
<i>xmit</i>	Queue\$token Returns: status	Closes off a line entered via byte mode by updating global give ptr to equal temporary give ptr. Status is either "normal" or "null".
<i>open\$line</i>	Queue\$token Returns: ptr	Opens up a line for byte oriented output. Ptr returned either points to the next byte or is zero indicating an empty queue.
<i>next\$char</i>	Queue\$token Returns: ptr	Increments temporary take pointer. Ptr returned either points to next byte or is zero indicating an empty queue.
<i>receive</i>	Queue\$token Returns: status	Closes off a line retrieved in byte mode by updating global take pointer to equal temporary ptr. Status is either "normal" or "null".

Table 3
Support Routines

Primitive	Arguments	Usage
<i>get\$status</i>	Queue\$token Returns: status	Returns status of queue. Possible values are "normal", "empty", "full" and "null".
<i>set\$interrupt</i>	Queue\$token, type Returns: status	Generates a slave → master or master → slave interrupt. Type code 0 is illegal and codes 8H → 0FH are reserved for use by the primitives.
<i>set\$handler</i>	Queue\$token, handler\$adr Returns: status	Inserts address into vector table used for handling interrupts described above.
<i>s\$init</i>	none	Called from slave software to initialize.
<i>m\$init</i>	none	Called from master software to initialize.

software is expecting EBCDIC. The routine on the slave can very easily perform the necessary code conversion before stuffing the data into a queue.

Sample Slave Software

Given the existence of the primitive routines the applications routines on the slave and master can deal with the specific duties of each device. The following paragraphs revisit the code from application example 2, first for the slave and then for the master. Full code listings for these programs can be found in Appendix D.

The flowchart for the terminal input handler resident on the iSBC 544 board is shown in Figure 28. Support is provided for deleting characters (Rubout), deleting lines (control-X), pausing and resuming output (control-S and control-Q) and terminating lines (escape and carriage return). The sections of code reproduced below use this terminal input handler to present an example of the use of the data transfer primitives to enter and edit a line of input from a terminal. The byte variable *value* is based on the address variable *value\$ptr* which is assigned by calls to the primitives. The routine *var\$inp* inputs and returns a data byte from an I/O port specified by a calling parameter. This is necessary since the particular USART to be serviced is determined by reading the 8259A in-service register.

```

/* case 1; rubout; delete char */
do;
  new$ptr=back$space(token);
  if new$ptr=length$ptr then
    dummy=echo(token+1,.(bell),1);
  else
    do;
      dummy=echo(token+1,.(BS,SP,BS),3);
      ptr=new$ptr;
      count=count-1;
    end;
  end;
end;

```

Following this, the byte input is checked to see if it is a control character and if so a block within a DO CASE statement is executed. As an example of one of these blocks, if the character input was a RUBOUT the code sequence below is executed. The *back\$space* primitive is called and a temporary pointer is returned to a location in the queue. A check is made to determine if the line was empty and, if so, a bell is echoed to signal the operator. If the pointer returned did not indicate

an invalid RUBOUT the real pointer is assigned the value of the temporary pointer and a backspace, space, backspace is echoed to delete the previous character on the screen. Lastly, the character count for the current line is decremented.

```

VALUE$PTR=NEXT$SPACE(Queue$NUMBER);
VALUE=VAR$INP(USART$DATA$PORT(NUM));

```

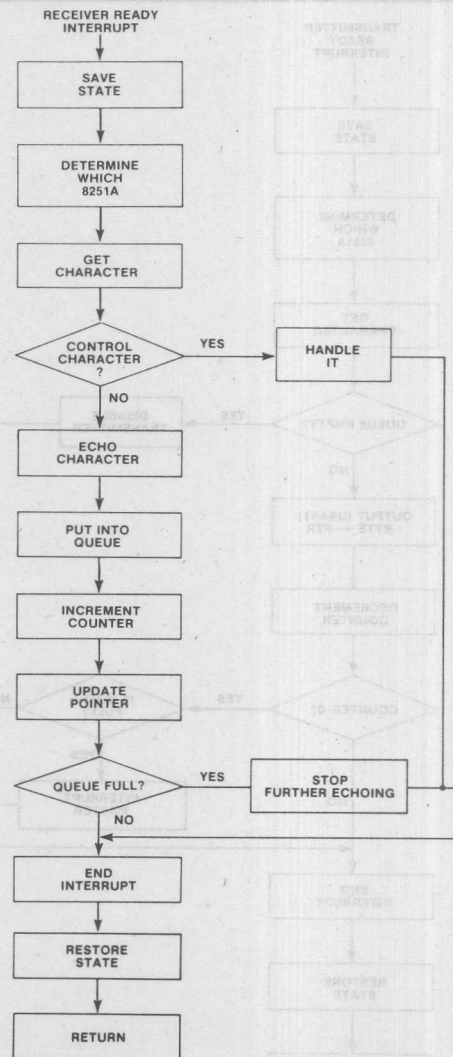


Figure 28. Flow Chart for Terminal Input Handler

In order to facilitate retrieval of the proper amount of information on the master side, the first byte of each message is defined to contain the number of characters in the message. Thus, when the master routine needs a line of input he uses the first byte as a count to retrieve the full line. The requirement for type-ahead is met by this mechanism since the number of lines in the

queue at a given time is limited only by the length of the queue. When a full line of input is finished, the terminal input handler generates a slave to master interrupt to signal the master routine who may be waiting for this event.

The flowchart for a minimal terminal output handler is shown in Figure 29. Upon receipt of a

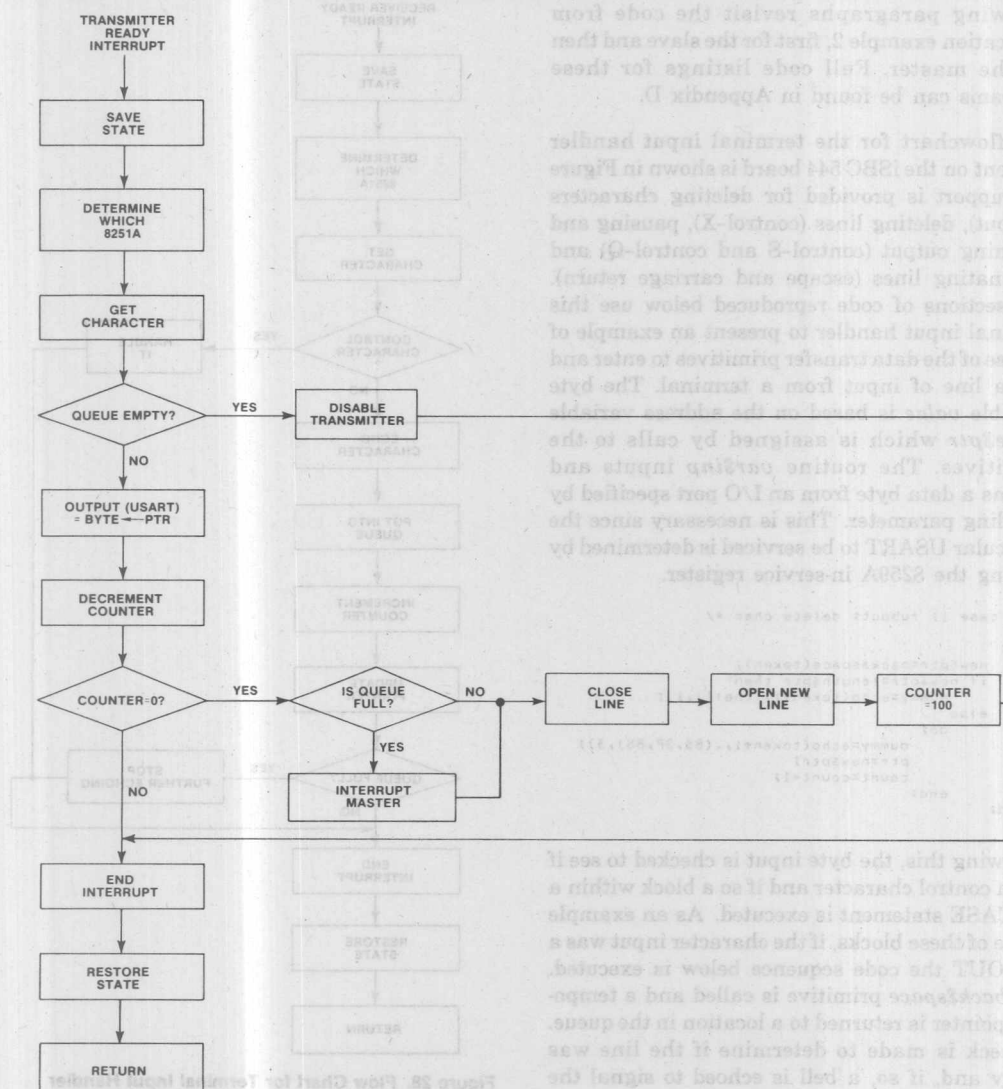


Figure 29. Flow Chart for Terminal Output Handler

transmitter ready interrupt the output handler requests a character from the appropriate queue. If one is available it is output to the USART. If the queue is empty, the transmitter is disabled. Whenever the master routine sends a line into the queue it will generate an interrupt to signal the slave handler and the transmitter will be reenabled. A line is opened via a call to *open\$line* and it is kept open until 100 characters have been retrieved via calls to *next\$byte*. At this time the line is closed by a call to *receive* making the space available to be reused. After this, a new call to *open\$line* starts the process over again. If the call to *get\$status* shows that the queue was full prior to the call to *receive*, an interrupt is sent to the master to reawaken any routine that may have been waiting for room in the queue to become available.

Sample Master Software

The RMX/80 handler for the master single board computer that will communicate with the software on the iSBC 544 board is diagrammed in Figure 30. In addition, the RMX/80 message used to convey information to the handler is shown on the right. The full software diagram is illustrated in Figure 31.

The input driver tasks execute a reentrant routine that services a request exchange that is specified in an initialization block that is unique to each of the input tasks. The necessary information is extracted from the request message and the *get\$line* primitive is called upon to get a line of input from the queue. If the call to *get\$line* for the length byte is unsuccessful the input task waits at

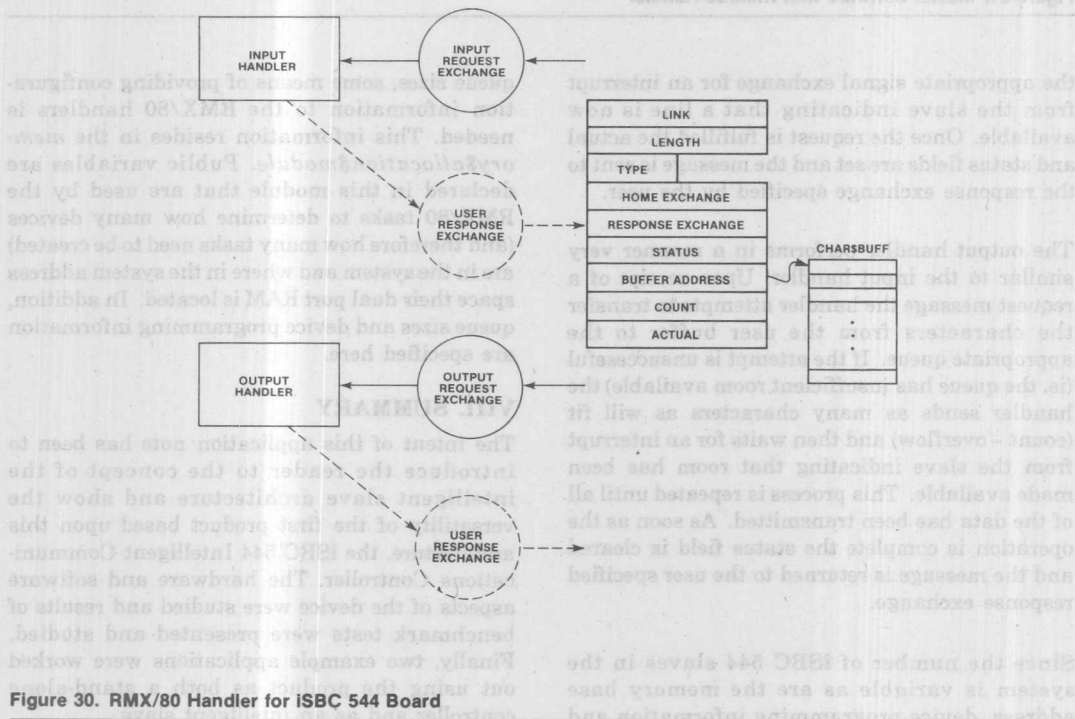


Figure 30. RMX/80 Handler for iSBC 544 Board

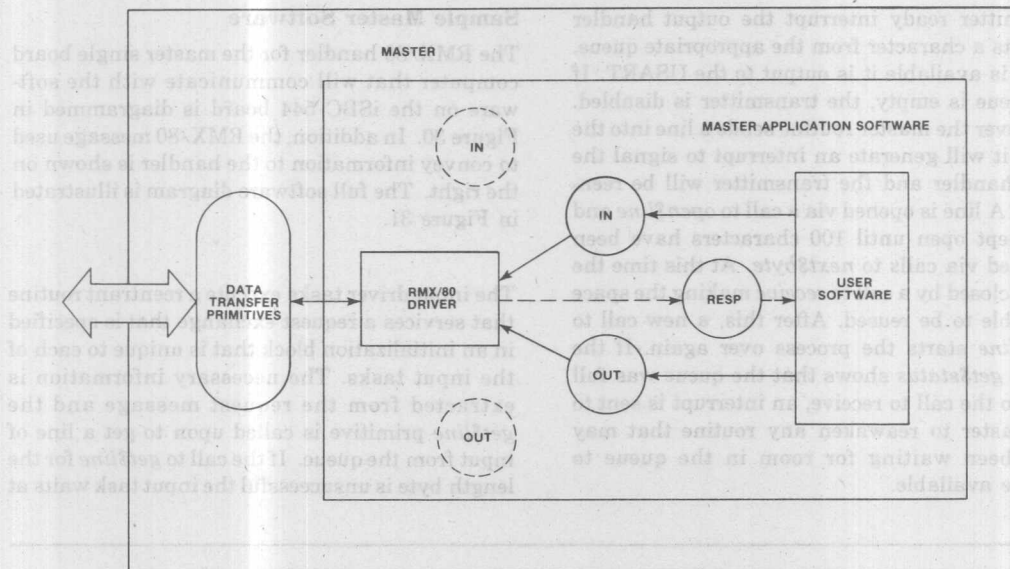


Figure 31. Master Software with RMX/80 Handler

the appropriate signal exchange for an interrupt from the slave indicating that a line is now available. Once the request is fulfilled the actual and status fields are set and the message is sent to the response exchange specified by the user.

The output handler performs in a manner very similar to the input handler. Upon receipt of a request message the handler attempts to transfer the characters from the user buffer to the appropriate queue. If the attempt is unsuccessful (ie. the queue has insufficient room available) the handler sends as many characters as will fit (count - overflow) and then waits for an interrupt from the slave indicating that room has been made available. This process is repeated until all of the data has been transmitted. As soon as the operation is complete the status field is cleared and the message is returned to the user specified response exchange.

Since the number of iSBC 544 slaves in the system is variable as are the memory base address, device programming information and

queue sizes, some means of providing configuration information to the RMX/80 handlers is needed. This information resides in the *memory\$allocation\$module*. Public variables are declared in this module that are used by the RMX/80 tasks to determine how many devices (and therefore how many tasks need to be created) are in the system and where in the system address space their dual port RAM is located. In addition, queue sizes and device programming information are specified here.

VIII. SUMMARY

The intent of this application note has been to introduce the reader to the concept of the intelligent slave architecture and show the versatility of the first product based upon this architecture, the iSBC 544 Intelligent Communications Controller. The hardware and software aspects of the device were studied and results of benchmark tests were presented and studied. Finally, two example applications were worked out using the product as both a stand-alone controller and as an intelligent slave.

The bottom line is that the iSBC 544 controller, due to the advanced architecture around which it is designed, can be the means to the end for any application that requires communication. The dual nature of the controller provides the full power of a single board computer to the small application while the large system can make use

of the fully programmable intelligent slave to free the CPU for complicated processing duties.

I would like to extend my gratitude to Dave Jurasek for the work on the throughput testing and to Jack Tyler Inman for aid in the design of the system software.

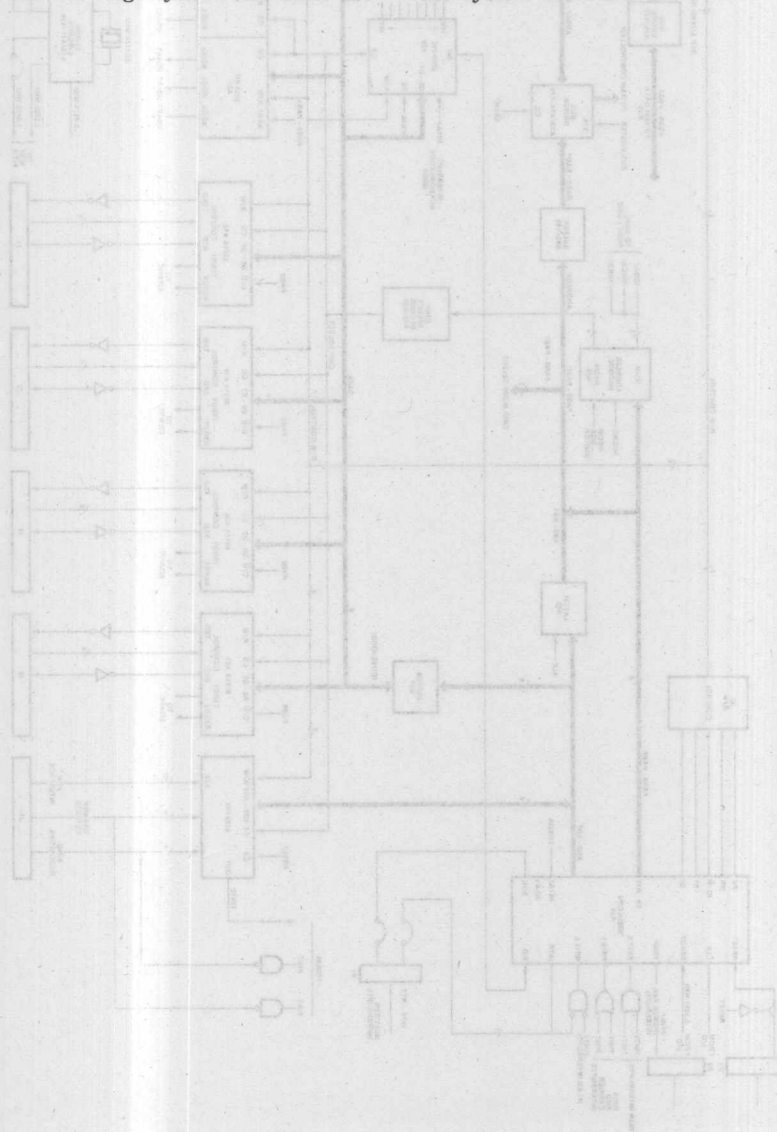
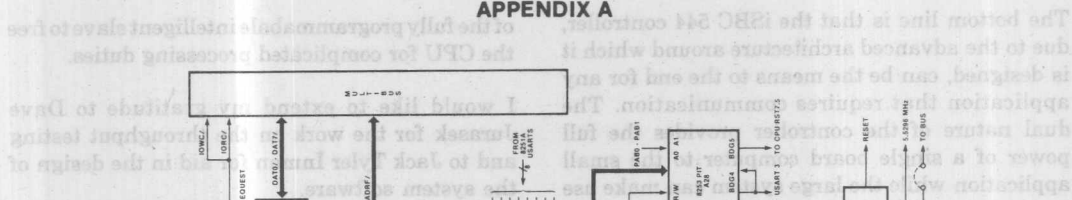


Figure A-1. iSBC 544 Input/Output and Interrupt Block Diagram

[illegible]

APPENDIX A (Continued)

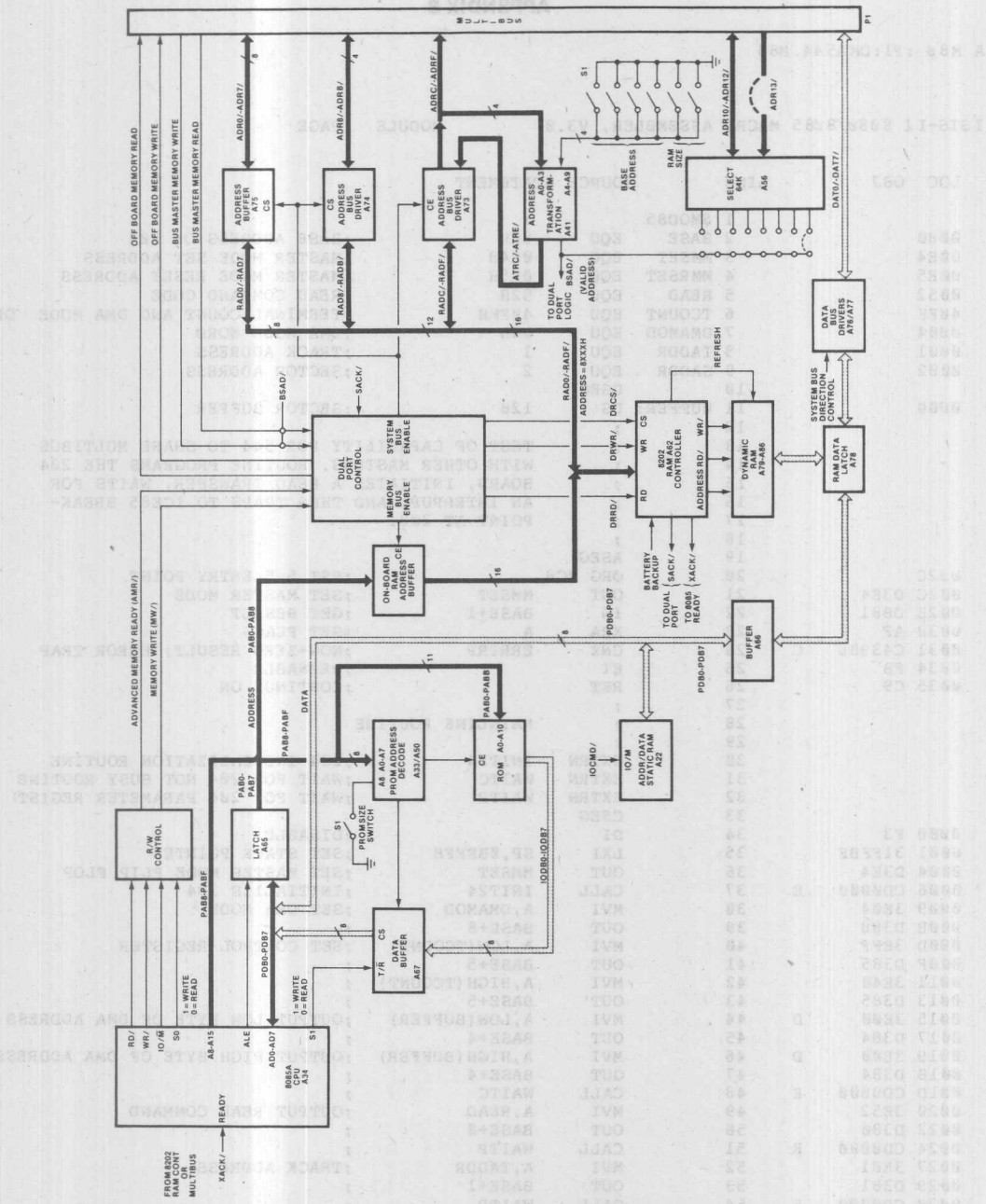


Figure A-2. iSBC 544 Memory Block Diagram

APPENDIX B

A M80 :FL:DMA544.M80

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0

MODULE PAGE 1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MOD85
0080		2	BASE EQU 80H ;BASE ADDRESS OF 204
00E4		3	MMSET EQU 0E4H ;MASTER MODE SET ADDRESS
00E5		4	MMRSET EQU 0E5H ;MASTER MODE RESET ADDRESS
0052		5	READ EQU 52H ;READ COMMAND CODE
40FF		6	TCOUNT EQU 40FFH ;TERMINAL COUNT AND DMA MODE 'OF
0004		7	DMAMOD EQU 04H ;DMA MODE WORD
0001		8	TADDR EQU 1 ;TRACK ADDRESS
0002		9	SADDR EQU 2 ;SECTOR ADDRESS
		10	DSEG
0000		11	BUFFER: DS 128 ;SECTOR BUFFER
		12	;
		13	;
		14	;
		15	TEST OF CAPABILITY FOR 544 TO SHARE MULTIBUS
		16	WITH OTHER MASTERS. ROUTINE PROGRAMS THE 204
		17	BOARD, INITIATES A READ TRANSFER, WAITS FOR
		18	AN INTERRUPT AND THEN TRAPS TO ICE85 BREAK-
		19	POINT AT 200.
		20	ASEG
002C		21	ORG 2CH ;RST 5.5 ENTRY POINT
002C D3E4		22	OUT MMSET ;SET MASTER MODE
002E DB81		23	IN BASE+1 ;GET RESULT
0030 AF		24	XRA A ;SET FLAGS
0031 C43900	C	25	CNZ ERRTRP ;NON-ZERO RESULT; ERROR TRAP
0034 FB		26	EI ;REENABLE
0035 C9		27	RET ;CONTINUE ON
		28	;
		29	MAINLINE ROUTINE
		30	EXTRN INIT24 ;204 INITIALIZATION ROUTINE
		31	EXTRN WAITC ;WAIT FOR 204 NOT BUSY ROUTINE
		32	EXTRN WAITP ;WAIT FOR 204 PARAMETER REGISTER
		33	CSEG
0000 F3		34	DI ;DISABLE
0001 31FFBF		35	LXI SP,0BFFFH ;SET STACK POINTER
0004 D3E4		36	OUT MMSET ;SET MASTER MODE FLIP FLOP
0006 CD0000	E	37	CALL INIT24 ;INITIALIZE 204
0009 3E04		38	MVI A,DMAMOD ;SET DMA MODE
000B D388		39	OUT BASE+8 ;
000D 3EFF		40	MVI A,LOW(TCOUNT) ;SET CONTROL REGISTER
000F D385		41	OUT BASE+5 ;
0011 3E40		42	MVI A,HIGH(TCOUNT) ;
0013 D385		43	OUT BASE+5 ;
0015 3E00	D	44	MVI A,LOW(BUFFER) ;OUTPUT LOW BYTE OF DMA ADDRESS
0017 D384		45	OUT BASE+4 ;
0019 3E00	D	46	MVI A,HIGH(BUFFER) ;OUTPUT HIGH BYTE OF DMA ADDRESS
001B D384		47	OUT BASE+4 ;
001D CD0000	E	48	CALL WAITC ;
0020 3E52		49	MVI A,READ ;OUTPUT READ COMMAND
0022 D380		50	OUT BASE+0 ;
0024 CD0000	E	51	CALL WAITP ;
0027 3E01		52	MVI A,TADDR ;TRACK ADDRESS
0029 D381		53	OUT BASE+1 ;
002B CD0000	E	54	CALL WAITP ;
002E 3E02		55	MVI A,SADDR ;SECTOR ADDRESS
0030 D381		56	OUT BASE+1 ;
0032 D3E5		57	OUT MMRSET ;RESET MASTER MODE FLIP/FLOP

APPENDIX B (Continued)

0034 FB	58	EI		;ENABLE
0035 76	59	HLT		;AND HALT ; WAIT FOR INTEF
0036 C30002	60	JMP	200H	;TRAP TO ICE85 BREAKPOINT AT 200
	61	ERRTRP:		;ERROR TRAP
0039 76	62	HLT		;FOR NOW
	63	END		

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

INIT24 E 0000	WAITC E 0000	WAITP E 0000			
USER SYMBOLS					
BASE A 0080	BUFFER D 0000	DMAMOD A 0004	ERRTRP C 0039	INIT24 E 0000	MF
READ A 0052	SADDR A 0002	TADDR A 0001	TCOUNT A 40FF	WAITC E 0000	WAI

APPENDIX B (Continued)

A M80 :F1:INIT24.M80

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0

MODULE PAGE 1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MOD85
0080		2	BASE EQU 80H ;BASE ADDRESS OF 204
00E4		3	MMSET EQU 0E4H ;MASTER MODE SET ADDRESS
00E5		4	MMRSET EQU 0E5H ;MASTER MODE RESET ADDRESS
0069		5	SEEK EQU 69H ;SEEK COMMAND
0035		6	SPECFY EQU 35H ;"SPECIFY" COMMAND CODE
0010		7	BADTRL EQU 10H ;SPECIFY BAD TRACKS SURFACE 1
0013		8	BADTR2 EQU 18H ;SPECIFY BAD TRACKS SURFACE 2
00FF		9	NOBAD EQU 0FFH ;NO BAD TRACKS
00FF		10	CTADDR EQU 0FFH ;CURRENT TRACK ADDRESS NOT KNOWN
000D		11	CHARS EQU 0DH ;SPECIFY DRIVE CHARACTERISTICS
0008		12	SETTLE EQU 08H ;HEAD SETTLE TIME(SA800)
0008		13	STEP EQU 08H ;STEP RATE
0009		14	LOAD EQU 09H ;HEAD LOAD TIME
4000		15	TCOUNT EQU 4000H ;TERMINAL COUNT AND DMA MODE OF
0004		16	DMAMOD EQU 04H ;DMA MODE WORD
0080		17	BUSY EQU 80H ;204 BUSY MASK
0020		18	PARFUL EQU 20H ;204 PARAMETER REGISTER FULL AS
0010		19	RESFUL EQU 10H ;204 RESULT BYTE FULL MASK
		20	;
		21	;
		22	;
		23	;
		24	;
		25	CSEG
		26	PUBLIC INIT24 ;ENTRY POINT
		27	PUBLIC WAITC ;WILL BE USED EXTERNALLY
		28	PUBLIC WAITP ;
		29	INIT24:
0000 F3		30	DI ;DISABLE
0001 3E0E		31	MVI A,0EH ;ENABLE 5.5 INTERRUPT
0003 30		32	SIM ;
0004 D3E4		33	OUT MMSET ;SET MASTER MODE FLIP FLOP
0006 D38F		34	OUT BASE+15 ;RESET INTERFACE
0008 3E01		35	MVI A,1 ;RESET 204
000A D382		36	OUT BASE+2 ;
000C AF		37	XRA A ;
000D D382		38	OUT BASE+2 ;
000F CD9900	C	39	CALL WAITC ;WAIT TILL COMMAND WRITE VAL7
0012 3E35		40	MVI A,SPECFY ;OUTPUT "SPECIFY" COMMAND
0014 D380		41	OUT BASE+0 ;
0016 CDA100	C	42	CALL WAITP ;WAIT TILL PARAMETER WRITE V7 II
0019 3E0D		43	MVI A,CHARS ;SPECIFYING DRIVE CHARACTERISTIC
001B D381		44	OUT BASE+1 ;
001D CDA100	C	45	CALL WAITP ;
0020 3E08		46	MVI A,STEP ;OUTPUT STEP RATE
0022 D381		47	OUT BASE+1 ;
0024 CDA100	C	48	CALL WAITP ;
0027 3E08		49	MVI A,SETTLE ;OUTPUT HEAD SETTLE TIME
0029 D381		50	OUT BASE+1 ;
002B CDA100	C	51	CALL WAITP ;
002E 3E09		52	MVI A,LOAD ;OUTPUT HEAD LOAD TIME
0030 D381		53	OUT BASE+1 ;
0032 CD9900	C	54	CALL WAITC ;

APPENDIX B (Continued)

0035	3E35		55	MVI	A,SPECIFY	;SPECIFY BAD TRACKS
0037	D380		56	OUT	BASE+0	;
0039	CDAl00	C	57	CALL	WAITP	;
003C	3E10		58	MVI	A,BADTR1	;BAD TRACKS FOR SURFACE 1
003E	D381		59	OUT	BASE+1	;
0040	CDAl00	C	59	CALL	WAITP	;
0043	3EFF		61	MVI	A,NOBAD	;FIRST TRACK
0045	D381		62	OUT	BASE+1	;
0047	CDAl00	C	63	CALL	WAITP	;
004A	3EFF		64	MVI	A,NOBAD	;SECOND BAD TRACK
004C	D381		65	OUT	BASE+1	;
004E	CDAl00	C	66	CALL	WAITP	;
0051	3EFF		67	MVI	A,CTADDR	;CURRENT TRACK ADDRESS (NOT F'OW
0053	D381		68	OUT	BASE+1	;
0055	CD9900	C	69	CALL	WAITC	;
0058	3E35		70	MVI	A,SPECIFY	;
005A	D380		71	OUT	BASE+0	;
005C	CDAl00	C	72	CALL	WAITP	;
005F	3E18		73	MVI	A,BADTR2	;SURFACE 2
0061	D381		74	OUT	BASE+1	;
0063	CDAl00	C	75	CALL	WAITP	;
0066	3EFF		76	MVI	A,NOBAD	;FIRST TRACK
0068	D381		77	OUT	BASE+1	;
006A	CDAl00	C	78	CALL	WAITP	;
006D	3EFF		79	MVI	A,NOBAD	;SECOND TRACK
006F	D381		80	OUT	BASE+1	;
0071	CDAl00	C	81	CALL	WAITP	;
0074	3EFF		82	MVI	A,CTADDR	;CURRENT TRACK ADDRESS (NOT F'OW
0076	D381		83	OUT	BASE+1	;
0078	CD9900	C	84	CALL	WAITC	;
007B	3E69		85	MVI	A,SEEK	;SEEK TO TRACK 0
007D	D380		86	OUT	BASE+0	;
007F	CDAl00	C	87	CALL	WAITP	;
0082	3E00		88	MVI	A,0	;
0084	D381		89	OUT	BASE+1	;
0086	D3E5		90	OUT	MMRSET	;GO TO SLEEP WHILE 204 DOES IT
0088	FB		91	EI		;ENABLE INTERRUPTS
0089	76		92	HLT		;SLEEP
008A	F3		93	DI		;DISABLE
008B	3E04		94	MVI	A,DMAMOD	;SET DMA MODE
008D	D388		95	OUT	BASE+8	;
008F	3E00		96	MVI	A,LOW(TCOUNT)	;SET CONTROL REGISTER
0091	D385		97	OUT	BASE+5	;
0093	3E40		98	MVI	A,HIGH(TCOUNT)	;
0095	D385		99	OUT	BASE+5	;
0097	FB		100	EI		;
0098	C9		101	RET		;RETURN
			102	;		
			103	;	WAITC AND WAITP ROUTINES	
			104	;		
0099	DB80		105	WAITC: IN	BASE+0	;GET STATUS BYTE
009B	E680		106	ANI	BUSY	;BUSY?
009D	C29900	C	107	JNZ	WAITC	;YES,LOOP
00A0	C9		108	RET		;NO,RETURN
			109	;		
00A1	DB80		110	WAITP: IN	BASE+0	;GET STATUS REGISTER
00A3	E620		111	ANI	PARFUL	;PARAMETER BUFFER FULL?
00A5	C2A100	C	112	JNZ	WAITP	;YES,LOOP
00A8	C9		113	RET		;NO,RETURN
			114	END		

PUBLIC SYMBOLS
INIT24 C 0000

WAITC C 0099

WAITP C 00A1

APPENDIX B (Continued)

EXTERNAL SYMBOLS

USER SYMBOLS

BADTR1 A 0010

```
INIT24 C 0000
```

BADTR2 A 0018

LOAD A 0009

SETTLE A 0008

BASE A 0080

MMRSET A 00E5

SPECFY A 0035

BUSY A 0080

MMSET A 00E4

STEP A 0008

CHARS A 000D

NOBAD A 00FF

TCOUNT A 4000

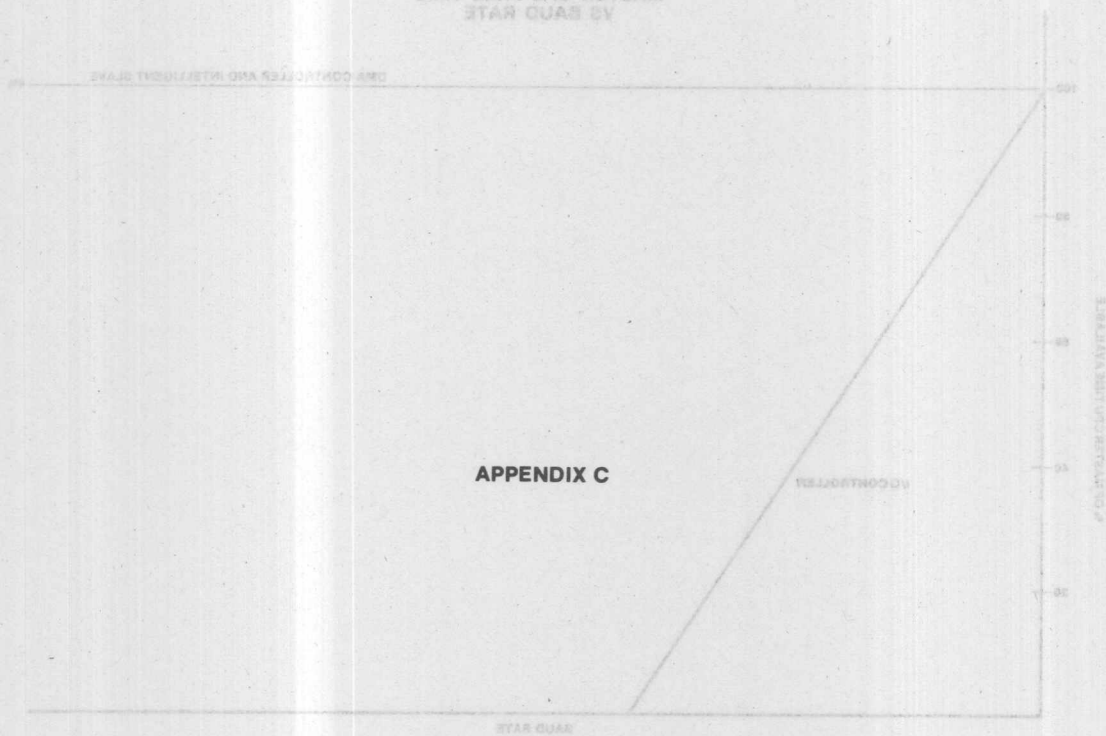
CTA

AF

WAI

APPENDIX C (Continued)

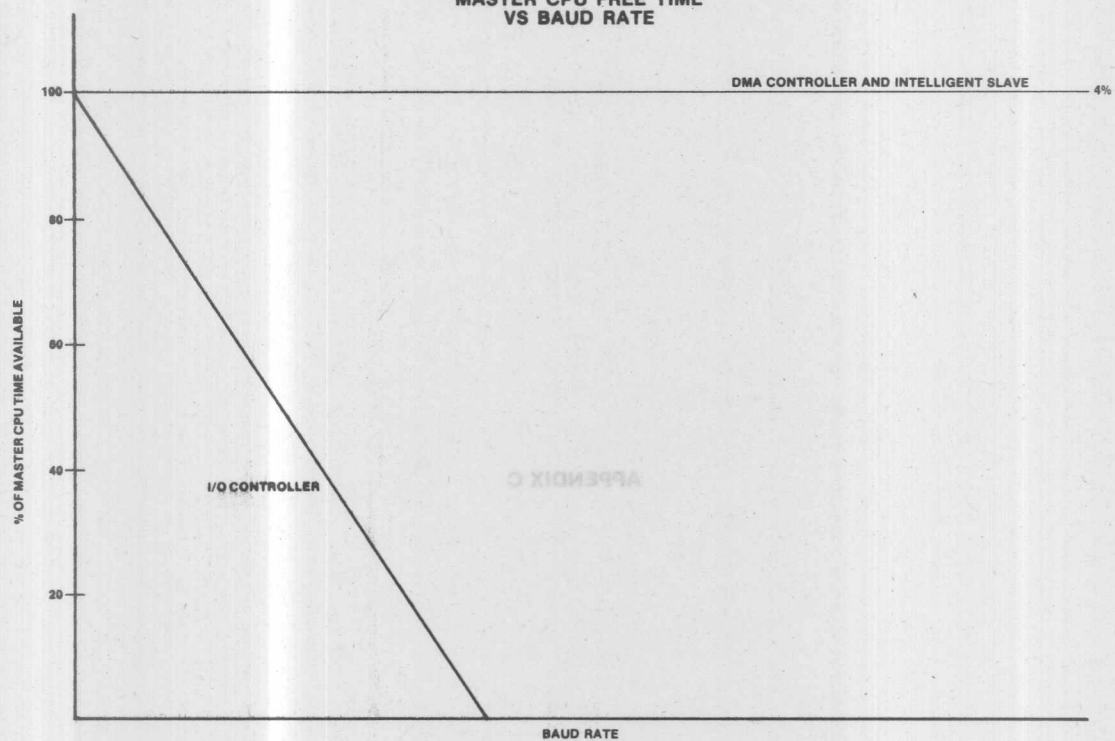
GRAPH 1
MASTER CPU FREE TIME
VS BAUD RATE



APPENDIX C

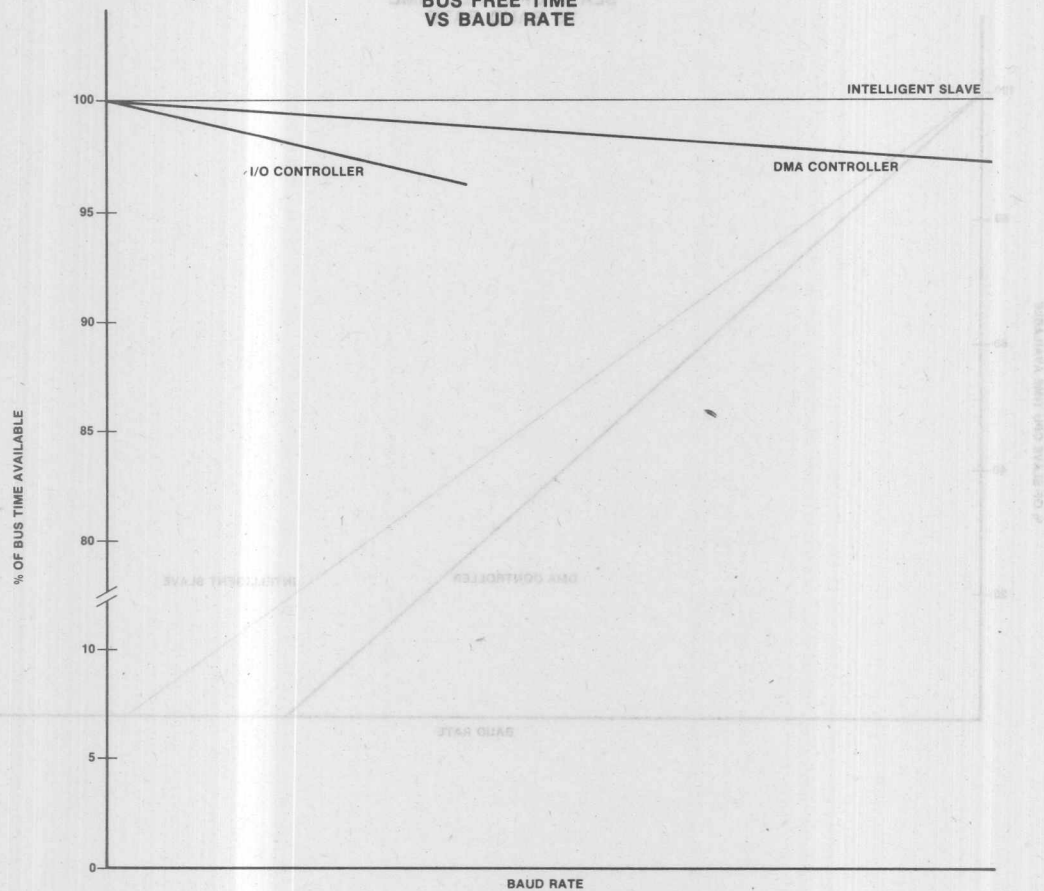
APPENDIX C (Continued)

GRAPH 1
MASTER CPU FREE TIME
VS BAUD RATE



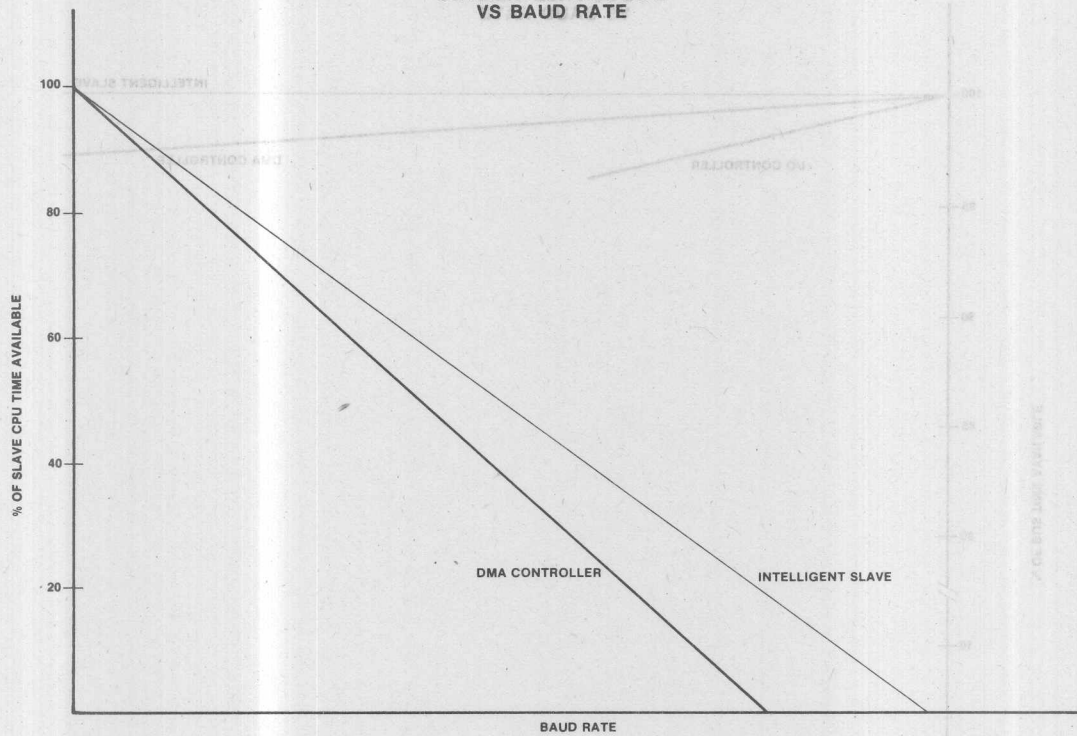
APPENDIX C (Continued)

GRAPH 2
BUS FREE TIME
VS BAUD RATE



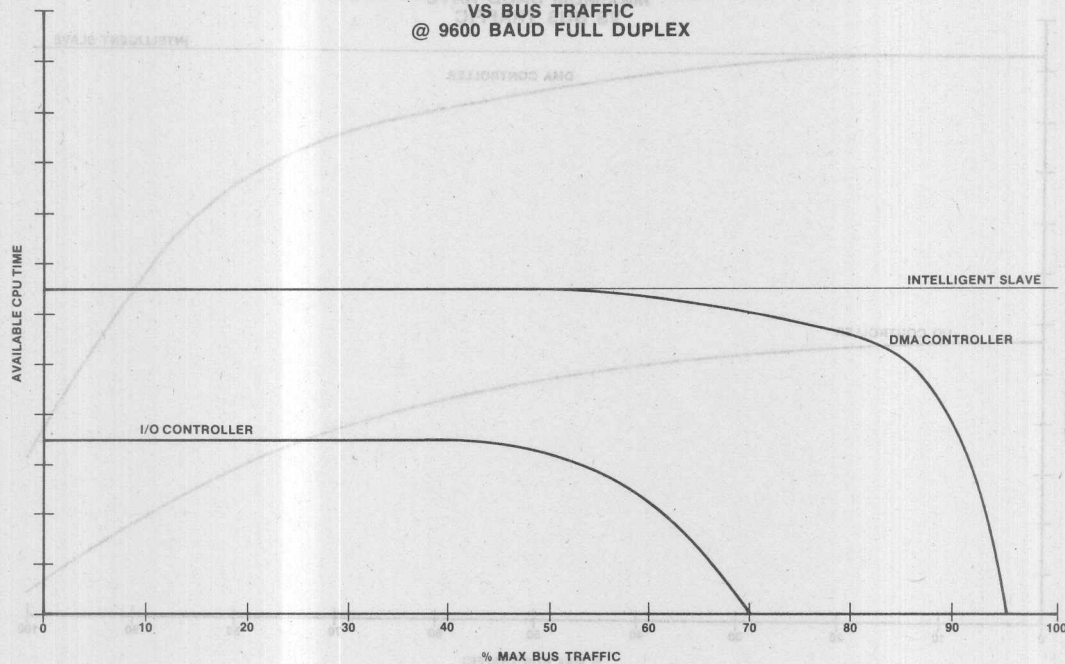
APPENDIX C (Continued)

GRAPH 3
SLAVE CPU FREE TIME
VS BAUD RATE



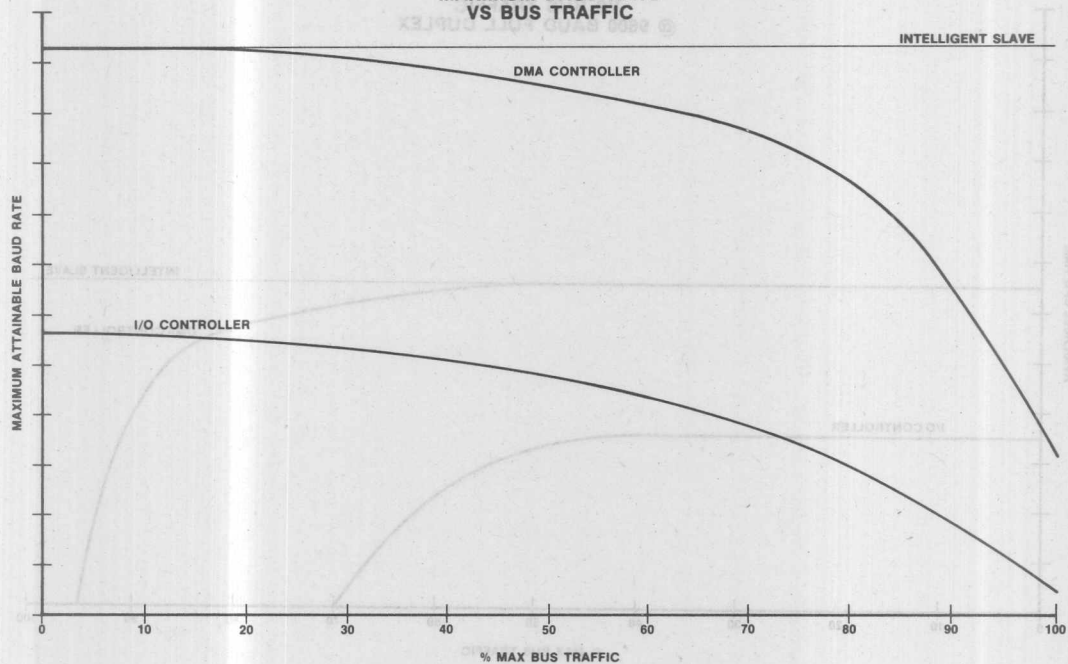
APPENDIX C (Continued)

GRAPH 4
COMMUNICATIONS PROCESSOR FREE TIME
VS BUS TRAFFIC
@ 9600 BAUD FULL DUPLEX



APPENDIX C (Continued)

GRAPH 5
MAXIMUM BAUD RATE
VS BUS TRAFFIC



(bun) APPENDIX D

PL/M-80 COMPILER SLAVE MAINLINE ROUTINE

PAGE 1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MAINLINE
OBJECT MODULE PLACED IN :F1:MAINLN.OBJ
COMPILER INVOKED BY: PLM80 :F1:MAINLN.PLM PRINT(:P5:MAINLN.LST) PAGewidth(78)

```
1      $title('slave mainline routine')
      main$line:
      DO;

      /*
      Mainline routine. Sets up stack$ptr, calls s$init to init-
      ialize queues, initializes some of the hardware, sets up the
      initial flag interrupt handlers, and then halts with interr
      - pts
      enabled allowing the rest of the system to operate totally
      in interrupt mode.
      */

      $nolist

13      1      initial$handler:  PROCEDURE EXTERNAL;
14      2      END initial$handler;

15      1      DECLARE
              command$word  LITERALLY  '4lh',
              port$a$8155 LITERALLY  '0e9h',
              command$8155  LITERALLY  '0e8h',
              mask$8259  LITERALLY  '0e7h',
              icw1$8259  LITERALLY  '0e6h',
              icw2$8259  LITERALLY  '0e7h',
              ocw3$8259  LITERALLY  '0e6h',
              read$isir  LITERALLY  '0bh',
              mask$word BYTE PUBLIC,
              port$a$value BYTE PUBLIC,
              stat  BYTE,
              i  BYTE;

16      1      output(icw1$8259)=0f6h;
17      1      output(icw2$8259)=0fh;
18      1      output(mask$8259),mask$word=0ffh;

19      1      CALL s$init;

      /* set up 8259 for ISR reads */

20      1      output(ocw3$8259)=read$isir;

21      1      output(command$8155)=command$word;
22      1      output(port$a$8155),port$a$value=0c0h;
23      1      DO i=0 TO 7;
24      2          stat=set$handler(i,.initial$handler);
```


APPENDIX D (Continued)

```

25 2      END;
26 1      DO WHILE 1;
27 2          HALT;
28 2      END;
29 1      END main$line;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 004DH      77D
VARIABLE AREA SIZE  = 0004H      4D
MAXIMUM STACK SIZE  = 0002H      2D
72 LINES READ
0 PROGRAM ERROR(S)

```

```

13 1      DECLARE
14 2      COMMAND$WORD LITERALLY 'AIN'
15 1      COMMAND$8125 LITERALLY 'begn'
16 1      COMMAND$8125 LITERALLY 'end'
17 1      MASK$3259 LITERALLY 'begn'
18 1      IOW$83259 LITERALLY 'begn'
19 1      IOW$83259 LITERALLY 'begn'
20 1      IOW$83259 LITERALLY 'begn'
21 1      IOW$83259 LITERALLY 'begn'
22 1      IOW$83259 LITERALLY 'begn'
23 1      IOW$83259 LITERALLY 'begn'
24 1      IOW$83259 LITERALLY 'begn'
25 1      IOW$83259 LITERALLY 'begn'
26 1      IOW$83259 LITERALLY 'begn'
27 1      IOW$83259 LITERALLY 'begn'
28 1      IOW$83259 LITERALLY 'begn'
29 1      IOW$83259 LITERALLY 'begn'
30 1      IOW$83259 LITERALLY 'begn'
31 1      IOW$83259 LITERALLY 'begn'
32 1      IOW$83259 LITERALLY 'begn'
33 1      IOW$83259 LITERALLY 'begn'
34 1      IOW$83259 LITERALLY 'begn'
35 1      IOW$83259 LITERALLY 'begn'
36 1      IOW$83259 LITERALLY 'begn'
37 1      IOW$83259 LITERALLY 'begn'
38 1      IOW$83259 LITERALLY 'begn'
39 1      IOW$83259 LITERALLY 'begn'
40 1      IOW$83259 LITERALLY 'begn'
41 1      IOW$83259 LITERALLY 'begn'
42 1      IOW$83259 LITERALLY 'begn'
43 1      IOW$83259 LITERALLY 'begn'
44 1      IOW$83259 LITERALLY 'begn'
45 1      IOW$83259 LITERALLY 'begn'
46 1      IOW$83259 LITERALLY 'begn'
47 1      IOW$83259 LITERALLY 'begn'
48 1      IOW$83259 LITERALLY 'begn'
49 1      IOW$83259 LITERALLY 'begn'
50 1      IOW$83259 LITERALLY 'begn'
51 1      IOW$83259 LITERALLY 'begn'
52 1      IOW$83259 LITERALLY 'begn'
53 1      IOW$83259 LITERALLY 'begn'
54 1      IOW$83259 LITERALLY 'begn'
55 1      IOW$83259 LITERALLY 'begn'
56 1      IOW$83259 LITERALLY 'begn'
57 1      IOW$83259 LITERALLY 'begn'
58 1      IOW$83259 LITERALLY 'begn'
59 1      IOW$83259 LITERALLY 'begn'
60 1      IOW$83259 LITERALLY 'begn'
61 1      IOW$83259 LITERALLY 'begn'
62 1      IOW$83259 LITERALLY 'begn'
63 1      IOW$83259 LITERALLY 'begn'
64 1      IOW$83259 LITERALLY 'begn'
65 1      IOW$83259 LITERALLY 'begn'
66 1      IOW$83259 LITERALLY 'begn'
67 1      IOW$83259 LITERALLY 'begn'
68 1      IOW$83259 LITERALLY 'begn'
69 1      IOW$83259 LITERALLY 'begn'
70 1      IOW$83259 LITERALLY 'begn'
71 1      IOW$83259 LITERALLY 'begn'
72 1      IOW$83259 LITERALLY 'begn'
73 1      IOW$83259 LITERALLY 'begn'
74 1      IOW$83259 LITERALLY 'begn'
75 1      IOW$83259 LITERALLY 'begn'
76 1      IOW$83259 LITERALLY 'begn'
77 1      IOW$83259 LITERALLY 'begn'
78 1      IOW$83259 LITERALLY 'begn'
79 1      IOW$83259 LITERALLY 'begn'
80 1      IOW$83259 LITERALLY 'begn'
81 1      IOW$83259 LITERALLY 'begn'
82 1      IOW$83259 LITERALLY 'begn'
83 1      IOW$83259 LITERALLY 'begn'
84 1      IOW$83259 LITERALLY 'begn'
85 1      IOW$83259 LITERALLY 'begn'
86 1      IOW$83259 LITERALLY 'begn'
87 1      IOW$83259 LITERALLY 'begn'
88 1      IOW$83259 LITERALLY 'begn'
89 1      IOW$83259 LITERALLY 'begn'
90 1      IOW$83259 LITERALLY 'begn'
91 1      IOW$83259 LITERALLY 'begn'
92 1      IOW$83259 LITERALLY 'begn'
93 1      IOW$83259 LITERALLY 'begn'
94 1      IOW$83259 LITERALLY 'begn'
95 1      IOW$83259 LITERALLY 'begn'
96 1      IOW$83259 LITERALLY 'begn'
97 1      IOW$83259 LITERALLY 'begn'
98 1      IOW$83259 LITERALLY 'begn'
99 1      IOW$83259 LITERALLY 'begn'
100 1      IOW$83259 LITERALLY 'begn'

```

APPENDIX D (Continued)

P /M-80 COMPILER SLAVE APPLICATION LEVEL SIGNAL HANDLE

PAGE 1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INITIALHANDLER
OBJECT MODULE PLACED IN :F1:FINTRT.OBJ
COMPILER INVOKED BY: PLM80 :F1:FINTRT.PLM PRINT(:F5:FINTRT.LST) PAGESWIDTH(78)

```

1      $title('slave application level signal handler')
      initial$handler:
      DO;

      /*
      Fields application level flag interrupts from the
      master. If the type=go$type the device attached to the queue
      specified is initialized with programming info sent into
      the queue by the master. If the type is data$available the
      specified transmitter is enabled unless a control$ pause
      is in effect.
      */

      $nolist

32 1  DECLARE
      no$pause LITERALLY '1',
      go$type LITERALLY '1',
      data$available LITERALLY '2',
      enable$xmit LITERALLY '1',
      reset LITERALLY '40h',
      timer$1$command$port LITERALLY '0dbh',
      timer$2$command$port LITERALLY '0dfh',
      mask$8259 LITERALLY '0e7h',
      mask$word BYTE EXTERNAL,
      mask (8) BYTE DATA(
          0fch,
          0fch,
          0f3h,
          0f3h,
          0cfh,
          0cfh,
          03fh,
          03fh),
      transmitter$state (8) BYTE PUBLIC,
      type BYTE,
      token BYTE,
      i BYTE,
      prog$info (5) BYTE,
      actual ADDRESS,
      usart$command$port (8) BYTE EXTERNAL,
      usart$state (8) BYTE PUBLIC,
      length$pointer (8) ADDRESS PUBLIC,
      pointer (8) ADDRESS PUBLIC,
      char$count (8) BYTE PUBLIC,
      timer$load$port (8) BYTE DATA(
          0d8h,

```

APPENDIX D (Continued)

```

PAGE 1
0d8h,
0d9h,
0d9h,
0dah,
0dah,
0dch,
0dch);

33 1 initial$handler: PROCEDURE (code) PUBLIC;
34 2 DECLARE code BYTE;
35 2 token=code AND 0fh;
36 2 type=shr(code,4);
37 2 IF type=go$type THEN
38 2 DO;
39 3 transmitter$state(token)=no$pause;
/* reset usart */
40 3 DO i=0 TO 3;
41 4 CALL varout(usart$command$port(token),0);
42 4 END;
43 3 CALL varout(usart$command$port(token),reset);
44 3 actual=get$line(token,.prog$info,5);
/* program the devices */
45 3 CALL varout(usart$command$port(token),prog$info(0));
46 3 CALL varout(usart$command$port(token),usart$state(token)
- :=prog$info(1));
47 3 IF token < 7 THEN
48 3 CALL varout(timer$1$command$port,prog$info(2));
ELSE
49 3 CALL varout(timer$2$command$port,prog$info(2));
50 3 CALL varout(timer$load$port(token),prog$info(3));
51 3 CALL varout(timer$load$port(token),prog$info(4));
/* open up the four input queues for data input */
52 3 length$pointer(token-1)=new$line(token-1);
53 3 pointer(token-1)=next$space(token-1);
54 3 char$count(token-1)=0;
55 3 output(mask$8259),mask$word=mask$word AND mask(token);
56 3 END;
ELSE
57 2 IF (type=data$available) AND (transmitter$state(token)=no$pa
- use) THEN
58 2 DO;
59 3 usart$state(token)=usart$state(token) OR enable$xmit;
60 3 CALL varout(usart$command$port(token),usart$state(token)
- );
61 3 END;

```

APPENDIX D (Continued)

```

63 2      RETURN;
64 1      END initial$handler;

```

MODULE INFORMATION:

```
CODE AREA SIZE      = 0182H      386D
VARIABLE AREA SIZE  = 0043H      67D
MAXIMUM STACK SIZE  = 0004H      4D
154 LINES READ
0 PROGRAM ERROR(S)
```


APPENDIX D (Continued)

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INPUTHANDLER
 OBJECT MODULE PLACED IN :F1:INHDLR.OBJ
 COMPILER INVOKED BY: PLM80 :F1:INHDLR.PLM PRINT(:F5:INHDLR.LST) PAGEWIDTH(78)

```

1      $nointvector title('slave terminal input handler')
      input$handler:
          DO;

/*
544 resident interrupt service routine. After receiver
ready interrupt the 8259 In Service Register(ISR) is
read to determine which device is requesting service.
The character is read in and placed in the appropriate
queue. A check is made for break characters and appropriate
action is taken if any are found. When an endline character
is encountered the length byte is filled in ( it was left
vacant when the line was started) and the xmit primitive is
called to update the global queue pointer to permit access
to the line. At this time the master is signalled to signify
that a new line is available for processing.

```

*/

\$nolist

```

34      1      DECLARE
          control$x      LITERALLY      '18H',
          control$s      LITERALLY      '13H',
          control$q      LITERALLY      '11H',
          rubout          LITERALLY      '7FH',
          escape          LITERALLY      '13H',
          CR              LITERALLY      '0DH',
          LF              LITERALLY      '0AH',
          BS              LITERALLY      '08H',
          SP              LITERALLY      '20H',
          bell            LITERALLY      '07H',
          ptr             LITERALLY      'pointer(token)',
          length$ptr      LITERALLY      'length$pointer(token)',
          count           LITERALLY      'char$count(token)',
          disable$xmit    LITERALLY      '0FEH',
          enable$xmit     LITERALLY      '01H',
          no$pause        LITERALLY      '1',
          pause           LITERALLY      '0',
          line$available  LITERALLY      '1',
          ocw2$8259       LITERALLY      '0E6H',
          ocw3$8259       LITERALLY      '0E6H',
          EOI             LITERALLY      '20H';

```

```

35      1      DECLARE
          value$ptr      ADDRESS,
          value          BASED value$ptr BYTE,
          line$length    BASED value$ptr BYTE,
          dummy          ADDRESS,
          ISR            BYTE,
          token          BYTE,

```

APPENDIX D (Continued)

```

stat BYTE,
new$ptr ADDRESS;

36 1 DECLARE
    pointer (8) ADDRESS EXTERNAL,
    length$pointer (8) ADDRESS EXTERNAL,
    char$count (8) BYTE EXTERNAL,
    usart$state (8) BYTE EXTERNAL,
    usart$command$port (8) BYTE EXTERNAL,
    usart$data$port (8) BYTE EXTERNAL,
    transmitter$state (8) BYTE EXTERNAL;

37 1 index: PROCEDURE (value) BYTE;
38 2 DECLARE value BYTE;

39 2 IF value=control$x THEN RETURN 0;
41 2 IF value=rubout THEN RETURN 1;
43 2 IF value=control$s THEN RETURN 2;
45 2 IF value=control$q THEN RETURN 3;
47 2 IF value=escape THEN RETURN 4;
49 2 IF value=CR THEN RETURN 5;
51 2 RETURN 6;
52 2 END;

53 1 echo: PROCEDURE (token,buf$ptr,num$char) ADDRESS;
54 2 DECLARE (buf$ptr,num$char,actual) ADDRESS,
    token BYTE;

55 2 actual=send$line(token,buf$ptr,num$char);
56 2 usart$state(token)=usart$state(token) OR enable$xmit;
57 2 CALL varout(usart$command$port(token),usart$state(token));
58 2 RETURN actual;
59 2 END;

60 1 delete$line: PROCEDURE;
61 2 length$ptr=new$line(token);
62 2 ptr=next$space(token);
63 2 count=0;
64 2 dummy=echo(token+1,(' ',CR,LF),3);
65 2 RETURN;
66 2 END;

67 1 end$line: PROCEDURE;
68 2 value$ptr=length$ptr;
69 2 line$length=count;
70 2 ptr=next$space(token);
71 2 stat=xmit(token);
72 2 length$ptr=new$line(token);
73 2 ptr=next$space(token);
74 2 count=0;
75 2 stat=set$s$interrupt(token,line$available);
76 2 RETURN;
77 2 END;

```

APPENDIX D (Continued)

```

78 1      in$hdlr:      PROCEDURE INTERRUPT 0 PUBLIC;
79 2          ISR=input(ocw3$8259);

80 2          token=6;

81 2      again:
          ISR=shl(ISR,2);

82 2          IF NOT carry THEN
83 2              DO;
84 3              IF token=0 THEN RETURN; /* no bits set */
                  ELSE
86 3                  DO;
87 4                      token=token-2;
88 4                      GOTO again;
89 4                  END;

90 3          END;
91 2          value$ptr=ptr;
92 2          value=varinp(usart$data$port(token)) AND 07fh;

93 2          DO CASE index(value);

                  /* case 0; control$x; delete line */
94 3              DO;
95 4                  CALL delete$line;
96 4              END;

                  /* case 1; rubout; delete char */
97 3              DO;
98 4                  new$ptr=back$space(token);
99 4                  IF new$ptr=length$ptr THEN
100 4                      dummy=echo(token+1,.(bell),1);
                  ELSE
101 4                      DO;
102 5                          dummy=echo(token+1,.(BS,SP,BS),3);
103 5                          ptr=new$ptr;
104 5                          count=count-1;
105 5                      END;
106 4                  END;

                  /* case 2; control$s; pause output */
107 3              DO;
108 4                  usart$state(token+1)=usart$state(token+1) AND disabl
-          e$xmit;
109 4                  CALL varout(usart$command$port(token+1),usart$state(
-          token+1));
110 4                  transmitter$state(token+1)=pause;
111 4              END;

                  /* case 3; control$q; resume output */
112 3              DO;
113 4                  usart$state(token+1)=usart$state(token+1) OR enable$

```

APPENDIX D (Continued)

```

PAGE 1
- xmit;
114 4 CALL varout(usart$command$port(token+1),usart$state(
- token+1));
115 4 transmitter$state(token+1)=no$pause;
116 4 END;
/* case 4; escape; terminate line */
117 3 DO;
118 4 dummy=echo(token+1,.( '#',CR,LF),3);
119 4 value=CR;
120 4 count=count+1;
121 4 CALL end$line;
122 4 END;
/* case 5; carriage return; terminate line */
123 3 DO;
124 4 dummy=echo(token+1,.(CR,LF),2);
125 4 count=count+1;
126 4 ptr=next$space(token);
127 4 value$ptr=ptr;
128 4 value=LF;
129 4 count=count+1;
130 4 CALL end$line;
131 4 END;
/* case 6; non-break character; stuff into queue */
132 3 DO;
133 4 dummy=echo(token+1,ptr,1);
134 4 ptr=next$space(token);
135 4 IF ptr=0 THEN CALL delete$line; /* full buffer */
137 4 ELSE count=count+1;
138 4 END;
139 3 END; /* of do case */
140 2 output(ocw3$8259)=EOI;
141 2 RETURN;
142 2 END;
143 1 END input$handler;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0398H      920D
VARIABLE AREA SIZE  = 0011H      17D
MAXIMUM STACK SIZE  = 0010H      16D
255 LINES READ
0 PROGRAM ERROR(S)

```


APPENDIX D (Continued)

P /M-80 COMPILER SLAVE CHARACTER OUTPUT HANDLER

PAGE 1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE OUTPUTHANDLER

OBJECT MODULE PLACED IN :F1:OUTHLR.OBJ

COMPILER INVOKED BY: PLM80 :F1:OUTHLR.PLM PRINT(:F5:OUTHLR.LST) PAGERWIDTH(78)

```

1      $nointvector title('slave character output handler')
      output$handler:
      DO;

```

```

/*
544 resident interrupt service routine. After transmitter
ready interrupt, 8259 In Service Register(ISR) is read to
determine which device is requesting service. A character
is requested from the appropriate queue and, if available,
is sent to the usart. If the queue is empty the transmitter
is disabled pending a signal from the master when more
characters are put into the queue.
*/

```

\$nolist

```

11 1  DECLARE
      ocw2$8259  LITERALLY  '0E6H',
      ocw3$8259  LITERALLY  '0E6H',
      disable$xmit  LITERALLY  '0FEH',
      true  LITERALLY  '0FFH',
      false  LITERALLY  '00H',
      EQI  LITERALLY  '0A0H';

```

```

12 1  DECLARE
      ISR BYTE,
      token  BYTE,
      actual  ADDRESS,
      value  BYTE;

```

```

13 1  DECLARE
      usart$state (8) BYTE EXTERNAL,
      usart$command$port (8) BYTE PUBLIC DATA(
          0D1H,
          0D1H,
          0D3H,
          0D3H,
          0D5H,
          0D5H,
          0D7H,
          0D7H),
      usart$data$port (8) BYTE PUBLIC DATA(
          0D0H,
          0D0H,
          0D2H,
          0D2H,
          0D4H,

```

APPENDIX D (Continued)

```

PAGE 1
00D4H,
00D6H,
00D6H);
14 1 out$hlr: PROCEDURE INTERRUPT,1,PUBLIC;
/* get active level number and use it to determine queue$token */
- /
15 2 ISR=input(ocw3$8259);
16 2 token=7;
17 2 again:
ISR=shl(ISR,1);
18 2 IF NOT carry THEN
19 2 DO;
20 3 IF token=1 THEN RETURN; /* no bits in ISR set */
ELSE
22 3 DO;
23 4 token=token-2;
24 4 ISR=shl(ISR,1);
25 4 GOTO again;
26 4 END;
27 3 END;
28 2 actual=get$line(token,.value,1);
29 2 IF actual=0 THEN
30 2 DO; /* empty queue. Disable transmitter */
31 3 usart$state(token)=usart$state(token) AND disabl
- e$xmit;
32 3 CALL varout(usart$command$port(token),usart$stat
- e(token));
33 3 END;
ELSE
34 2 CALL varout(usart$data$port(token),value);
35 2 output(ocw3$8259)=EOI;
36 2 RETURN;
37 2 END;
38 1 END output$handler;

```

MODULE INFORMATION:

```

CODE AREA SIZE = 00A4H 164D
VARIABLE AREA SIZE = 0005H 5D
MAXIMUM STACK SIZE = 000CH 12D
102 LINES READ
0 PROGRAM ERROR(S)

```

APPENDIX D (Continued)

PL/M-80 COMPILER

RMX/80-544 INITIALIZATION TASK

PAGE 1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INIT544
 OBJECT MODULE PLACED IN :F1:INIT54.OBJ
 COMPILER INVOKED BY: PLM80 :F1:INIT54.PLM PRINT(:F5:INIT54.LST) PAGERWIDTH(78)

```

1      $title('rmx/80-544 initialization task')
      init$544:
          DO;

/*
    Task code for 544 driver initialization task. Info
    from application supplied memory allocation block
    is accessed to set up queues and transfer device programming
    info to the slave board(s) and create the required
    service handler tasks.
*/

      $nolist

56     1      input$driver:  PROCEDURE EXTERNAL;
57     2      END input$driver;

58     1      output$driver: PROCEDURE EXTERNAL;
59     2      END output$driver;

60     1      signal: PROCEDURE EXTERNAL;
61     2      END signal;

62     1      DECLARE
              stack$size  LITERALLY  '256',
              go$type     LITERALLY  '1';

63     1      DECLARE
              ptr          ADDRESS,
              init$stable  BASED ptr STRUCTURE(
                  base$adr ADDRESS,
                  queue$token BYTE,
                  prog$info (5) BYTE),
              i            BYTE,
              overflow     ADDRESS,
              queue$init$stable (1) STRUCTURE(
                  base$adr ADDRESS,
                  queue$size (8) ADDRESS) EXTERNAL,
              initialization$stable (1) BYTE EXTERNAL,
              stat         BYTE,
              num$devices  BYTE EXTERNAL,
              num$boards  BYTE EXTERNAL,
              service$exchange$stable (1) ADDRESS EXTERNAL,
              signal$exchange$stable (1) ADDRESS EXTERNAL,
              service$exchanges (1) BYTE EXTERNAL,
              signal$exchanges (1) BYTE EXTERNAL,
              task$descriptors (1) BYTE EXTERNAL,

```

APPENDIX D (Continued)

```

stacks (1) BYTE EXTERNAL;
info$block (1) STRUCTURE(
    base$adr ADDRESS,
    queue$token BYTE,
    index BYTE) EXTERNAL;
rgactv ADDRESS EXTERNAL;

64 1 DECLARE
    rom$input$std static$task$descriptor DATA(
        'input',
        .input$driver,
        0, /* stack will be assigned individually */
        stack$size,
        200,
        0, /* tba */
        0, /* tba */
        rom$output$std static$task$descriptor DATA(
            'output',
            .output$driver,
            0,
            stack$size,
            201,
            0,
            0,
            input$hdlr$std static$task$descriptor,
            output$hdlr$std static$task$descriptor;

65 1 init$xch: PROCEDURE (xch$ptr);
    /* initializes expanded interrupt exchanges */

66 2 DECLARE xch$ptr ADDRESS,
    xch BASED xch$ptr int$exchange$descriptor;

67 2 xch.link=.xch.link;
68 2 xch.type=int$type;
69 2 xch.length=5;
70 2 RETURN;
71 2 END;

72 1 init$54: PROCEDURE PUBLIC;

73 2 DO i=0 TO num$boards-1;
74 3 CALL m$init(.queue$init$table(i));
75 3 END;

76 2 CALL move(size(rom$input$std),.rom$input$std,.input$hdlr$std
- );
77 2 CALL move(size(rom$output$std),.rom$output$std,.output$hdlr$
- std);
78 2 ptr=.initialization$table;

79 2 DO i=0 TO num$devices*2 BY 2;
/* send programming info to slave */
80 3 overflow=send$line(init$table.base$adr,init$table.queue$
- token,.init$table.prog$info,5);
81 3 stat=set$m$interrupt(init$table.base$adr,init$table.queu

```


APPENDIX D (Continued)

```

- e$token,go$type);
/* create service and signal exchanges */
82 3      CALL rqcxcx(service$exchange$table(i):=.service$exchange
- s+10*i);
83 3      CALL rqcxcx(service$exchange$table(i+1):=.service$exchan
- ges+10*(i+1));
84 3      CALL init$xch(.signal$exchanges+15*i);
85 3      CALL init$xch(.signal$exchanges+15*(i+1));
86 3      CALL rqcxcx(signal$exchange$table(i):=.signal$exchanges+
- 15*i);
87 3      CALL rqcxcx(signal$exchange$table(i+1):=.signal$exchange
- s+15*(i+1));

88 3      info$block(i).base$adr,
info$block(i+1).base$adr=init$table.base$adr;
89 3      info$block(i).queue$token=init$table.queue$token-1;
90 3      info$block(i+1).queue$token=init$table.queue$token;
91 3      info$block(i).index=i;
92 3      info$block(i+1).index=i+1;

93 3      input$hdlr$std.sp=.stacks+stack$size*i;
94 3      output$hdlr$std.sp=.stacks+stack$size*(i+1);
95 3      input$hdlr$std.exchange$address=.info$block(i);
96 3      output$hdlr$std.exchange$address=.info$block(i+1);
97 3      input$hdlr$std.task$ptr=.task$descriptors+20*i;
98 3      output$hdlr$std.task$ptr=.task$descriptors+20*(i+1);

99 3      CALL rqctsk(.input$hdlr$std);
100 3      CALL rqctsk(.output$hdlr$std);
101 3      ptr=ptr+8;
102 3      END;

103 2      CALL rqsetv(.signal,2);
104 2      CALL rqelvl(2);
105 2      CALL rqsusp(rqactv);
106 2      END; /* of task */

107 1      END init$544;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 02C3H      707D
VARIABLE AREA SIZE  = 002AH      42D
MAXIMUM STACK SIZE  = 0006H      6D
285 LINES READ
0 PROGRAM ERROR(S)

```

APPENDIX D (Continued)

P /M-80 COMPILER RMX/80-544 INITIALIZATION MODULE AND

PAGE 1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INITMODULE
OBJECT MODULE PLACED IN :F1:MAB.OBJ
COMPILER INVOKED BY: PLM80 :F1:MAB.PLM PRINT(:F5:MAB.LST) PAGESWIDTH(78)

```

1      $title('rmx/80-544 initialization module and memory allocation b
-      lock')
1      init$module:
        DO;

        /*
-      44      Initialization tables created and allocation of memory for 5
        handler done here.
        */

2      1      DECLARE
        number$of$devices LITERALLY '4',
        baud$rate$count$1 LITERALLY '32',
        baud$rate$count$h LITERALLY '00',
        usart$mode LITERALLY '4eh',
        usart$cmd LITERALLY '27h',
        ctr$0$mode LITERALLY '36h',
        ctr$1$mode LITERALLY '76h',
        ctr$2$mode LITERALLY '0b6h',
        ctr$3$mode LITERALLY '36h',
        num$devices BYTE PUBLIC DATA(number$of$devices-1),
        num$boards BYTE PUBLIC DATA(1),
        service$exchange$table (8) ADDRESS PUBLIC,
        signal$exchange$table (8) ADDRESS PUBLIC,
        signal$type (8) BYTE PUBLIC,
        service$exchanges (80) BYTE PUBLIC,
        signal$exchanges (120) BYTE PUBLIC,
        task$descriptors (160) BYTE PUBLIC,
        stacks (2048) BYTE PUBLIC,
        info$block (32) BYTE PUBLIC,
        queue$init$table (1) STRUCTURE(
            base$adr ADDRESS,
            queue$size (8) ADDRESS) PUBLIC DATA(
                0e000h,
                256,
                1765,
                256,
                1765,
                256,
                1765,
                256,
                1765,
                256,
                1765),
        base$table (1) ADDRESS PUBLIC DATA(
            0e000h),
        initialization$table (number$of$devices) STRUCTURE(
            base$adr ADDRESS,

```

APPENDIX D (Continued)

```

PAGE 1
queue$token BYTE,
prog$info (5) BYTE) PUBLIC DATA(
0e000h,
1,
usart$mode,
usart$cmd,
ctr$0$mode,
baud$rate$count$1,
baud$rate$count$h,
0e000h,
3,
usart$mode,
usart$cmd,
ctr$1$mode,
baud$rate$count$1,
baud$rate$count$h,
0e000h,
5,
usart$mode,
usart$cmd,
ctr$2$mode,
baud$rate$count$1,
baud$rate$count$h,
0e000h,
7,
usart$mode,
usart$cmd,
ctr$3$mode,
baud$rate$count$1,
baud$rate$count$h);
3 1 END init$module;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0036H      54D
VARIABLE AREA SIZE  = 09B0H      2480D
MAXIMUM STACK SIZE  = 0000H      0D
79 LINES READ
0 PROGRAM ERROR(S)

```

APPENDIX D (Continued)

P /M-80 COMPILER SLAVE->MASTER INTERRUPT HANDLER PAGE 1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SIGNALHANDLER
OBJECT MODULE PLACED IN :F1:SIGNAL.OBJ
COMPILER INVOKED BY: PLM80 :F1:SIGNAL.PLM PRINT(:F5:SIGNAL.LST) PAGEWIDTH(78)

```

1      $nointvector title('slave->master interrupt handler')
      signal$handler:
          DO;

          /*
          - d Fields all slave->master signals(interrupts) and calls rgisn
              with the proper signal exchange address.
          */

          $nolist

26      1      DECLARE
                  i      BYTE,
                  ptr      ADDRESS,
                  (flag BASED ptr) BYTE,
                  num$boards BYTE EXTERNAL,
                  num$devices BYTE EXTERNAL,
                  signal$type (1) BYTE EXTERNAL,
                  index      BYTE,
                  token      BYTE,
                  signal$exchange$table (1) ADDRESS EXTERNAL,
                  base$table (1) ADDRESS EXTERNAL;

27      1      signal: PROCEDURE INTERRUPT 2 PUBLIC;

          /* poll slave boards and find one generating interrupt */

28      2      i=0;

29      2      next:
                  ptr=base$table(i)+1;
30      2      IF flag=0 THEN
31      2          DO;
32      3              i=i+1;
33      3              IF i > num$boards THEN RETURN; /* erroneous signal */
          - /
35      3              ELSE GOTO next;
36      3      END;

          /* get queue token and use it to index into signal exchange tabl
          - e */

37      2      token=(flag AND 0fh);
38      2      index=4*i+token;

          /* if index is out of range don't attempt the isend */

```


APPENDIX D (Continued)

```

39 2      IF index <= num$devices THEN
40 2      DO;
41 3          CALL rquisnd(signal$exchange$stable(index));
42 3          signal$type(index)=shr(flag,4);
43 3      END;
      ELSE
44 2          CALL rquendi;

      /* zero flag to acknowledge interrupt */

45 2          flag=0;
46 2          RETURN;
47 2      END;

48 1      END signal$handler;

```

MODULE INFORMATION:

```
CODE AREA SIZE      = 008BH      139D
VARIABLE AREA SIZE  = 0005H      5D
MAXIMUM STACK SIZE  = 000AH      10D
110 LINES READ
0 PROGRAM ERROR(S)
```

APPENDIX D (Continued)

P /M-80 COMPILER RMX/80-544 INPUT SERVICE HANDLER PAGE 1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INPUTDRIVER,
OBJECT MODULE PLACED IN :F1:INPUT.OBJ
COMPILER INVOKED BY: PLM80 :F1:INPUT.PLM PRINT(:F5:INPUT.LST) PAGESWIDTH(78)

```

1      $title('rmx/80-544 input service handler')
      input$driver:
      DO;

      /*
      Master resident task code. Monitors service exchange
      and fills input requests by retrieving characters from
      the proper queue(board$base and device info is passed
      via default exchange field). By definition the first byte
      of a line of input contains the length of that line.
      This figure is used to retrieve the exact number of characte
      - rs
      available in a given line.
      */
      $nolist

27 1      DECLARE
          rgactv ADDRESS EXTERNAL,
          td BASED rgactv task$descriptor,
          service$exchange$table (1) ADDRESS EXTERNAL,
          signal$exchange$table (1) ADDRESS EXTERNAL;

28 1      input$driver: PROCEDURE REENTRANT PUBLIC;

29 2      DECLARE
          service$exchange ADDRESS,
          board$base ADDRESS,
          queue$token BYTE,
          signal$exchange ADDRESS,
          msg$ptr ADDRESS,
          msg BASED msg$ptr th$msg,
          actual ADDRESS,
          dummy ADDRESS,
          info$block$ptr ADDRESS,
          info$block BASED info$block$ptr STRUCTURE(
              base$adr ADDRESS,
              queue$token BYTE,
              index BYTE),
          num$char BYTE,
          stat BYTE;

      /* get info out of default field */

30 2      info$block$ptr=td.exchange$address; /* default exchange fiel
31 2      - d */
          service$exchange=service$exchange$table(info$block.index);
    
```

APPENDIX D (Continued)

```

32 2 board$base=info$block.base$adr;
33 2 queue$token=info$block.queue$token;
34 2 signal$exchange=signal$exchange$stable(info$block.index);
35 2 DO forever;

/* wait for request message */

36 3 msg$ptr=rqwait(service$exchange,0);

37 3 retry:
/* try to get line count out of queue */
actual=get$line(board$base,queue$token,.num$char,1);

/* if unsuccessful wait for signal and try again */

38 3 IF actual=0 THEN
39 3 DO;
40 4 dummy=rqwait(signal$exchange,0);
41 4 GOTO retry;
42 4 END;

/* if all okay get line */

43 3 actual=get$line(board$base,queue$token,msg.buffer$adr,num
- m$char);
44 3 msg.actual=actual;
45 3 msg.status=0;
46 3 CALL rgsend(msg.resp$ex,msg$ptr);
47 3 END; /* of do forever */

48 2 END; /* of task */

49 1 END input$driver;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 012CH 300D
VARIABLE AREA SIZE  = 0000H  0D
MAXIMUM STACK SIZE  = 0017H 23D
171 LINES READ
0 PROGRAM ERROR(S)

```

APPENDIX D (Continued)

P /M-80 COMPILER RMX/80-544 OUTPUT SERVICE HANDLER PAGE 1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE OUTPUTDRIVER
OBJECT MODULE PLACED IN :F1:OUTPUT.OBJ
COMPILER INVOKED BY: PLM80 :F1:OUTPUT.PLM PRINT(:F5:OUTPUT.LST) PAGEWIDTH(78)

```

1      $title('rmx/80-544 output service handler')
      output$driver:
      DO;

      /*
      Master resident task code. Monitors service exchange and
      fills output requests by stuffing characters into the appropriate
      queue. If insufficient room is available the task waits
      for 1 second and retries up to 100 times after which it
      signals a time out error. If the transmission completes
      successfully the slave is signalled to indicate that data is
      available.
      */
      $nolist

39  1      DECLARE
          data$available LITERALLY '2',
          time$out LITERALLY '1';

40  1      DECLARE
          rqactv ADDRESS EXTERNAL,
          (td BASED rqactv) task$descriptor,
          service$exchange$stable (1) ADDRESS EXTERNAL,
          signal$exchange$stable (1) ADDRESS EXTERNAL;

41  1      output$driver: PROCEDURE REENTRANT PUBLIC;

42  2      DECLARE
          service$exchange ADDRESS,
          signal$exchange ADDRESS,
          base$adr ADDRESS,
          queue$token BYTE,
          msg$ptr ADDRESS,
          msg BASED msg$ptr th$msg,
          tries$left BYTE,
          overflow ADDRESS,
          dummy ADDRESS,
          stat BYTE,
          info$block$ptr ADDRESS,
          info$block BASED info$block$ptr STRUCTURE(
              base$adr ADDRESS,
              queue$token BYTE,
              index BYTE);

```


APPENDIX D (Continued)

```

PAGE      /* initialize */
43  2      info$block$ptr=td.exchange$address;
44  2      service$exchange=service$exchange$table(info$block.index);
45  2      signal$exchange=signal$exchange$table(info$block.index);
46  2      base$adr=info$block.base$adr;
47  2      queue$token=info$block.queue$token;

48  2      DO forever;

          /* wait for request message */

49  3      msg$ptr=rqwait(service$exchange,0);
50  3      tries$left=100;
51  3      retry: overflow=send$line(base$adr,queue$token,msg.buffer$adr,m
          - sg.count);
52  3      IF overflow <> 0 THEN
53  3      DO;
54  4      dummy=rqwait(signal$exchange,20);
55  4      tries$left=tries$left-1;
56  4      IF tries$left > 0 THEN GOTO retry;
          ELSE
58  4      DO;
59  5      msg.status=time$out;
60  5      msg.actual=0;
61  5      GOTO quit;
62  5      END;
63  4      END;
64  3      msg.status=0;
65  3      stat=set$m$interrupt(base$adr,queue$token,data$available
          - );
66  3      msg.actual=msg.count;
67  3      quit: CALL rgsend(msg.resp$ex,msg$ptr);
68  3      END; /* of do forever */

69  2      END; /* of task */

70  1      END output$driver;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0159H 345D
VARIABLE AREA SIZE = 0000H 0D
MAXIMUM STACK SIZE = 0019H 25D
198 LINES READ
0 PROGRAM ERROR(S)

```

APPENDIX D (Continued)

A M80 :F1:CFG544.M80 PRINT(:F4:CFG544.LST) PAGEWIDTH(78) MACROFILE

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0

CFG544 PAGE 1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	NAME CFG544
		2	CSEG
		3	PUBLIC RQRATE
0000	0300	4	RQRATE: DW 8
		5	\$NOLIST
		127	\$LIST
		128	\$NOGEN
0000		129	NTASK SET 0
0000		130	NEXCH SET 0
		131	;
		132	; BUILD THE INITIAL TASK TABLE
		133	;
		134	; -----\ THIS TASK IS NECESSARY FOR THE 544 HANDLE
			R
		135	; -----/ IT CREATES EVERYTHING ELSE IT NEEDS.
		136	STD INIT54,200,1,0
		191	STD LINECH,64,130,0
		246	
		247	; ALLOCATE TASK DESCRIPTORS
		248	; GENTD
		249	
		253	; BUILD INITIAL EXCHANGE TABLE
		254	; XCHADR RESPEX
		255	; BUILD CREATE TABLE
		256	
		264	; CRTAB
		265	; END
		266	
		267	
		274	

PUBLIC SYMBOLS
RQRTB C 0026 RQRATE C 0000

EXTERNAL SYMBOLS
INIT54 E 0000 LINECH E 0000 RESPEX E 0000

USER SYMBOLS
CRTAB + 0000 GENTD + 0000 IET C 0024 INIT54 E 0000
INTXCH + 0000 ITT C 0002 LINECH E 0000 NEXCH A 0001
NTASK A 0002 PUBXCH + 0007 RESPEX E 0000 RQRTB C 0026
RQRATE C 0000 STD + 0000 TDBASE D 0108 XCH + 0005
XCHADR + 0002



APPLICATION NOTE

AP-149

APPENDIX D (Continued)

October 1982

Using the iSBC[®] 550 Ethernet* Communications Controller

Stephen J. Packer
OEM Microsystems
Applications Engineering

1 INTRODUCTION

With the reduced cost of computers brought about by the advances in silicon-based computer components, computers have become distributed throughout offices and factories. The decentralization of the computing facilities is far from complete, and one significant impediment is the exchange of information amongst these distributed computers.

Intercomputer communications can be divided into three groups: global, local, and bus communications.^[4] Global refers to those facilities which are capable of spanning large distances. Local communications span smaller distances, generally limited to one building or one small group of buildings, and provide higher performance than global networks. Bus communications span even smaller distances (within a single computer) and provide even more performance.

One intercomputer communication method, Ethernet, developed jointly by Digital Equipment Corporation, Intel Corporation and Xerox Corporation provides the facilities for the local communications group.^[13, 21] This communication method is exactly defined by the Ethernet Specification V1.0.^[11]

This Application Note provides information on Ethernet and the first Intel-supplied products for use of Ethernet. The note also addresses the software which must be supplied by the customer to effectively use the new Ethernet capabilities. Specifically the following topics are covered:

- A brief overview of the Ethernet Specification
- A discussion of performance characteristics of Ethernet
- Expected usage of Ethernet in the office and factory
- An overview of the iSBC 550 Ethernet Controller
- The measured characteristics of the iSBC 550 Ethernet Controller in the laboratory
- A test in an electrically noisy environment
- A description of an application which is typical of software to be provided by the customer

The reader will be able to assess the development effectiveness and cost of the use of the iSBC 550 Ethernet Controller in a system design.

Intel provides an implementation of the Ethernet Specification with the iSBC 550 Ethernet Communications Controller. This controller is a full-featured implementation of the specification and is effective both for evaluating Ethernet and for early introduction of Ethernet products. The controller can be integrated into MULTIBUS-based systems.

We chose the new System 86/330 microcomputer for integration with the iSBC 550 Ethernet Controller. This System is representative of the kinds of systems expected

on an Ethernet network. Further, the System 86/330 provided the tools necessary for the development of an Ethernet application.

The application chosen for demonstration of Ethernet fulfilled two requirements. First, it provided a typical and useful function. Second, it provided an example of multi-vendor interconnection using Ethernet. The result of creating the Ethernet application is a quantified measure of the effort necessary for multivendor interconnection.

The iSBC 550 Ethernet Controller is supported by the E-Driver. The E-Driver is software which provides a standard system I/O interface for the iRMX 86 Operating System. The E-Driver uses the iMMX 800 MULTIBUS Message Exchange software for communication with the Ethernet controller. A flexible and modular system can be constructed from the integration of standard Intel single board computers, a memory board, an iSBC 550 Ethernet Controller, the iMMX 800 MULTIBUS Message Exchange, the iRMX 86 Operating System, and the E-Driver.

2 THE ETHERNET* SPECIFICATION

The Ethernet Specification,^[11] developed as a joint effort by Digital Equipment Corporation, Intel Corporation and Xerox Corporation, is a result of earlier experimental work done by Xerox.^[2, 13, 20] This work is based on a need for a high-speed interconnect of large numbers of computers. Further, the network must be easy to reconfigure and immune to total network failure caused by single component failure. These ambitious goals led to an experimental network which has been operated successfully since 1972.

Ease of reconfigurability of a network is important in office environments and those factory environments which continually add, delete or reconfigure their shop floor plan. Non-technical users will remove network devices and transport them to another location. Such activities should not disrupt the remaining part of the network.

One important goal often forgotten by implementors of networks is simplicity. The implementations should be easy to understand and devoid of marginally useful features. In achieving this goal, the network must remain flexible and not impose restrictions that reflect only a narrow application focus. Ethernet achieved this goal with the Carrier Sense Multiple Access/Collision Detect (CSMA/CD) access method.

The CSMA/CD access method is conceptually simple, the exact same interface exists in all nodes, and the algorithm readily lends itself to VLSI solutions. These characteristics are being recognized as exemplified by the effort to standardize Ethernet.

The Ethernet Specification defines Ethernet exactly to ensure that hardware and software developed by various

manufacturers will be compatible with those developed by all others. This is a significant advancement in data communications where few standards have been written and none implemented exactly.

Multivendor acceptance and implementation of Ethernet provides a significant first step to an open system implementation of a communications facility. However, additional software not specified by the standard is required before a total system solution is possible.

Most networks are structured into layers to reduce the complexity of implementation. The most successful structuring is that proposed in the ISO Reference Model for Open System Interconnect.^[15] This model is commonly used for implementing networks because it partitions the solution to network problems into understandable entities. Each entity is both comprehensive and cohesive.

The model is briefly described as follows:

ISO Model Layer Primary Function

- | | |
|----------------|---|
| 7. Application | <ul style="list-style-type: none"> • All possible uses of lower layers, virtual file, virtual terminal, file transfer, job control, etc. |
|----------------|---|

- | | |
|-----------------|---|
| 6. Presentation | <ul style="list-style-type: none"> • Procedure call structure, code set, data structure |
| 5. Session | <ul style="list-style-type: none"> • Binding of distributed functions |
| 4. Transport | <ul style="list-style-type: none"> • End-to-end transport of data, flow control, reliability, multiplexing |
| 3. Network | <ul style="list-style-type: none"> • Intermediate nodes, multiplexing |
| 2. Data Link | <ul style="list-style-type: none"> • Recovery from high error rate media |
| 1. Physical | <ul style="list-style-type: none"> • Electrical, mechanical, and procedural characteristics of media. |

The Ethernet Specification is organized as an architectural design, (see Figure 2-1) corresponding to the lowest layers of the ISO Model for Open Systems Interconnect.^[15, 26] The specification defines two layers: data link layer and physical layer. The data link layer defines a communication facility for the sending or receiving of data frames which is independent of the actual physical medium. The physical layer provides a physical channel through a coaxial cable medium.

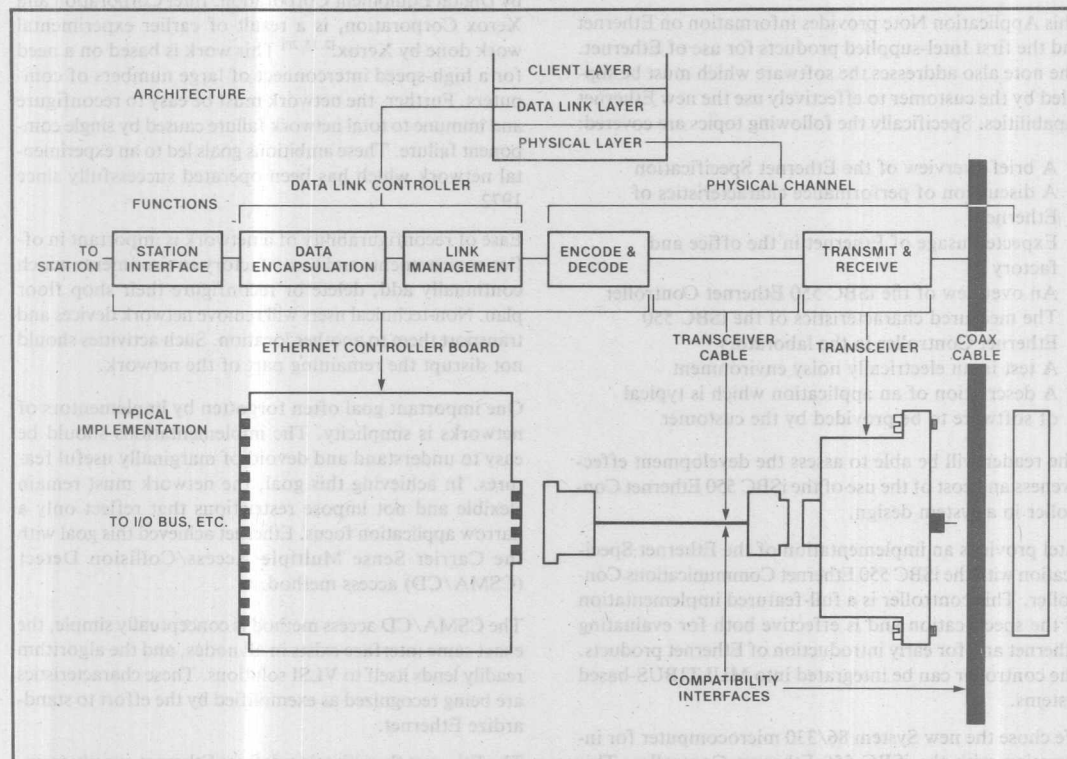


Figure 2-1. Ethernet* Architecture and Typical Implementation

The client layer is defined as higher levels of network architecture not defined by the Ethernet Standard. This layer may consist of the additional layers specified by the ISO Model. The client layer uses the data link and physical layers to transmit and receive frames or error status.

2.1 Data Link Layer

The data link layer provides two main functions. First, it provides data encapsulation which handles frame addressing, frame-check-sequence generation and verification, and frame delimitation. Second, the data link layer provides link management which handles the access to the physical channel and handles resolution of conflicts.

2.1.1 FRAME FORMAT

The frame format of the data link layer is shown in Figure 2-2. It consists of a destination and source address, a type, data, and a frame-checking sequence. The destination address field of an Ethernet frame may be either a physical address uniquely assigned to each particular station on the Ethernet or a multicast address. The multicast address is associated with a group of logically related stations. A station may be a member of more than one multicast group. One predefined multicast address, the broadcast address, is associated with all stations on a given Ethernet. The station's physical address is distinct from the physical address on any other station on any Ethernet.

Station and multicast addresses are assigned and administered by Xerox for a nominal fee.^[1] The source address is the physical address of the sending station and is provided for use by the client layer.

The type field is reserved for use by the client layers to identify a particular client layer protocol. The type field values are also administered by Xerox Corporation for a nominal fee. It is not interpreted by the data link layer. The data field consists of a sequence of n octets, where $46 < n < 1500$. An octet is a set of 8 bits. Any arbitrary sequence of octet values may appear in the fields. The frame-checking sequence is a 32-bit cyclic redundancy code computed as a function of the contents of the destination, source, type and data field.

2.1.2 FRAME TRANSMISSION AND COLLISION RESOLUTION

The data link layer monitors the carrier sense signal provided by the physical layer for channel busy. When the channel is busy, any transmission is deferred until the channel is no longer busy. The data link layer continues to defer for 9.6 microseconds to ensure a standard inter-frame spacing which provides an interframe recovery time for other data link controllers.

It is possible for two stations to begin transmission at nearly the same time. This contention for the network can occur only within the short time required for a single station

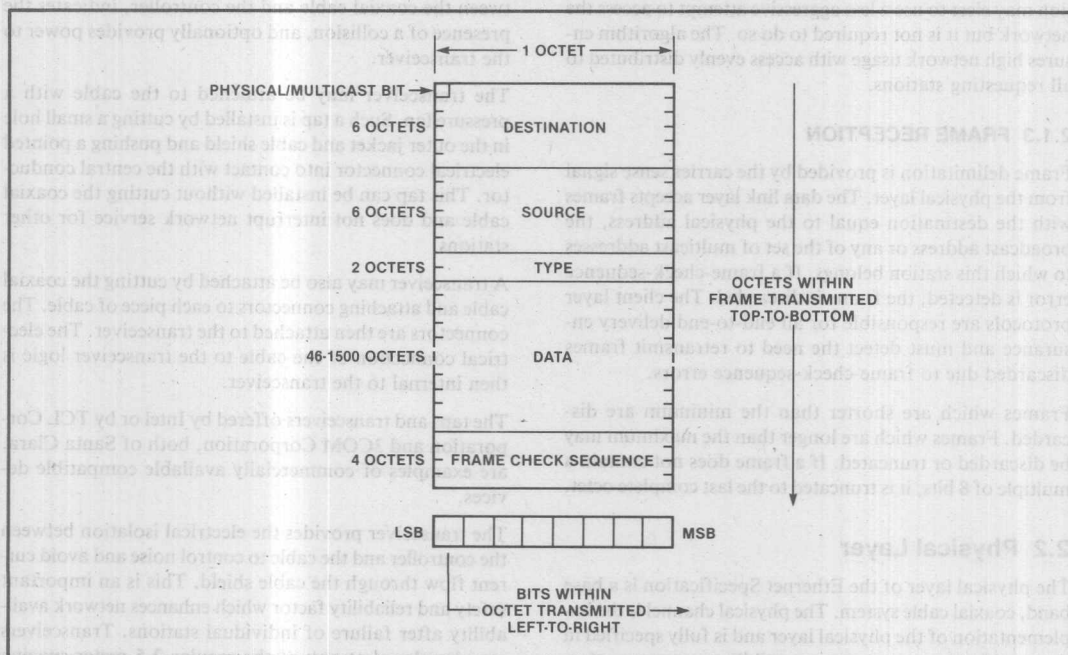


Figure 2-2. Data Link Layer Frame Format

to acquire the network. (All other stations detect carrier sense.) The data link layer detects collisions by monitoring the collision-detect signal provided by the physical layer. Upon detection of a collision, transmission continues for at least 32 but not more than 48 bits to ensure collision detection by all other stations. The time to acquire the network is dependent on the propagation speed of the electrical signal in the coaxial cable (0.77c worst case where c is the speed of light), the propagation speed in transceiver cable (0.65c worst case), the lengths of the cables, and the number of repeaters. The worst-case time to acquire the network is 45 microseconds.

When a transmission has terminated due to a collision, it is retransmitted until successful or until 16 attempts have been made. All attempts to retransmit a frame are completed before a subsequent frame is transmitted. The retransmission of frames experiencing a collision is controlled by selection of a random delay interval. The delay algorithm provides a graceful and fair access method for a station during peak loading.

The algorithm is graceful because it reduces the throughput experienced by a station in small increments. Any one station will not experience a long delay where it is excluded from requesting network access. Thus a station may be perceived to run slower but it does not unexpectedly stop.

The algorithm is fair since it permits all stations on the network to bid for network access with equal priority. A station may elect to use a less aggressive attempt to access the network but it is not required to do so. The algorithm ensures high network usage with access evenly distributed to all requesting stations.

2.1.3 FRAME RECEPTION

Frame delimitation is provided by the carrier sense signal from the physical layer. The data link layer accepts frames with the destination equal to the physical address, the broadcast address or any of the set of multicast addresses to which this station belongs. If a frame-check-sequence error is detected, the frame is discarded. The client layer protocols are responsible for all end-to-end delivery assurance and must detect the need to retransmit frames discarded due to frame-check-sequence errors.

Frames which are shorter than the minimum are discarded. Frames which are longer than the maximum may be discarded or truncated. If a frame does not contain a multiple of 8 bits, it is truncated to the last complete octet.

2.2 Physical Layer

The physical layer of the Ethernet Specification is a base band, coaxial cable system. The physical channel is the implementation of the physical layer and is fully specified in the specification to ensure compatibility across manufacturers. The channel consists of a coaxial cable, a trans-

ceiver, and data encoder/decoder. Repeaters are used to reach the maximum allowable distances and to provide topological flexibility. Three configurations of Ethernet are given in Figure 2-3, 2-4 and 2-5.

2.2.1 THE COAXIAL CABLE

The cable specifications are intended to ensure proper operation of the Ethernet in its varied topologies. The cable is a robust, double-shield coaxial cable which is suitable for installation in office or factory environments such as dropped ceilings, raised floors and cable troughs. The cable was chosen to provide acceptable reflection characteristics caused by transceivers or connectors and to provide adequate noise shielding. BELDON AWM STYLE 1478 ETHERNET COAX IE is one commercially available cable that meets the Ethernet Specification.

The cable is terminated at both ends to provide matching termination impedance to reduce reflections from the ends of the cable. The network will not operate without these terminations. Connectors are defined by the standard for joining more than one segment into a single, electrically continuous segment.

2.2.2 THE TRANSCEIVER

The data link controller is coupled to the cable by the transceiver. The transceiver interface transfers data between the coaxial cable and the controller, indicates the presence of a collision, and optionally provides power to the transceiver.

The transceiver may be attached to the cable with a pressure tap. Such a tap is installed by cutting a small hole in the outer jacket and cable shield and pushing a pointed electrical connector into contact with the central conductor. This tap can be installed without cutting the coaxial cable and does not interrupt network service for other stations.

A transceiver may also be attached by cutting the coaxial cable and attaching connectors to each piece of cable. The connectors are then attached to the transceiver. The electrical connection of the cable to the transceiver logic is then internal to the transceiver.

The taps and transceivers offered by Intel or by TCL Corporation and 3COM Corporation, both of Santa Clara, are examples of commercially available compatible devices.

The transceiver provides the electrical isolation between the controller and the cable to control noise and avoid current flow through the cable shield. This is an important safety and reliability factor which enhances network availability after failure of individual stations. Transceivers may be placed at any of the regular 2.5 meter spacing marks on the coaxial cable and up to 100 transceivers may

exist on any one cable segment. These restrictions are intended to reduce insertion reflections.

If the transceivers are to be separately packaged from the data link controller, a cable of up to 50 meters may be used. The cable consists of four-stranded, twisted pair conductors with a shield and insulating jacket. The connectors are also specified in the Standard.

2.2.3 ENCODER/DECODER

The clock and data are encoded into a single, self-clocking serial data stream. The encoder drives the transmit pair of transceiver cables using Manchester Encoding. The

data is preceded by a preamble to allow channel circuitry throughout the network to reach a steady state before receiving valid input. The decoder removes the preamble and provides output clock and data from the transceiver cable receiver pair.

2.2.4 REPEATERS

The topology of a network may be extended by adding repeaters which propagate signals from one cable segment to another. A maximum of 2 repeaters may be in the signal path between any two transceivers. Only one path may exist between any two transceivers. The repeaters must propagate both data signals and collision detect signals.

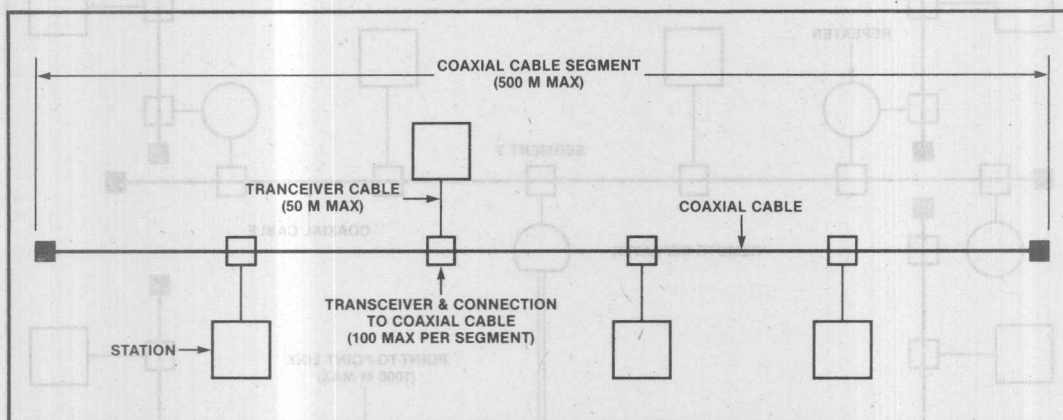


Figure 2-3. A Typical Small-scale Configuration

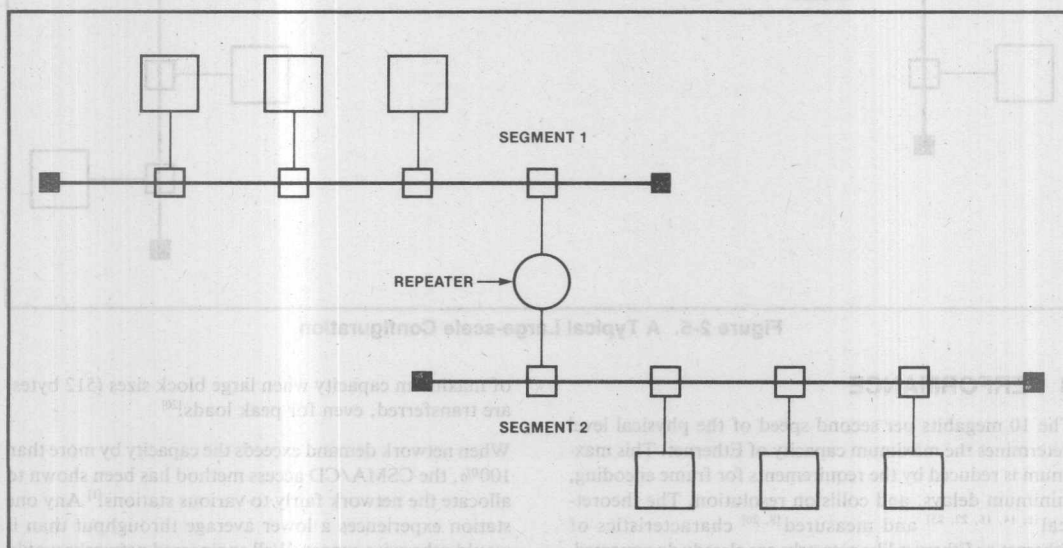


Figure 2-4. A Typical Medium-scale Configuration

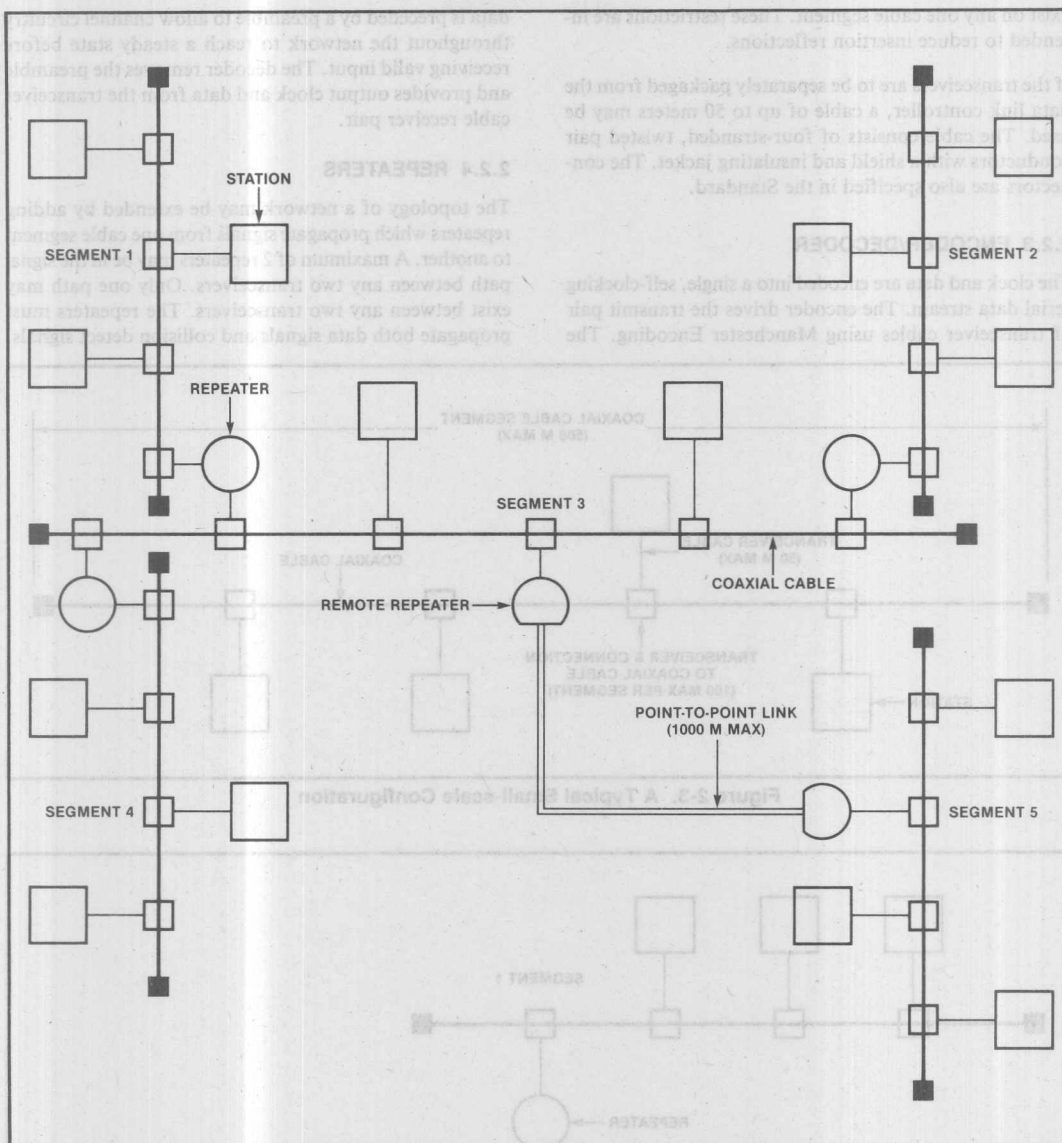


Figure 2-5. A Typical Large-scale Configuration

3 PERFORMANCE

The 10 megabits per second speed of the physical level determines the maximum capacity of Ethernet. This maximum is reduced by the requirements for frame encoding, minimum delays, and collision resolution. The theoretical^[1, 14, 16, 22, 25] and measured^[8, 20] characteristics of Ethernet or Ethernet-like networks are already documented. The utilization of the network can be kept at levels of 95%

of maximum capacity when large block sizes (512 bytes) are transferred, even for peak loads.^[20]

When network demand exceeds the capacity by more than 100%, the CSMA/CD access method has been shown to allocate the network fairly to various stations.^[1] Any one station experiences a lower average throughput than it would otherwise expect. Well engineered networks would be configured to expect an average total load well below

the maximum (40-80%). Peak loading would then be of a bursty nature and easily handled by Ethernet.

The selection of Ethernet for a particular application would be in response to requirements for high-speed communications between computers. One reason for computer interconnection is to share high-speed (or high-cost) devices such as laser printers or streamer tapes. Another reason is the sharing of common data bases, such as files contained on a disk system. The data rates for these devices determine the interface requirements and network throughput rates.

The following performance numbers are representative of devices commonly considered for attachment to Ethernet:

Device	Average Throughput
1. Winchester Disk	800K baud
2. Streamer Tape	800K baud
3. Printer	500K baud
4. Floppy Disk	250K baud
5. CRT (workstation)	10K baud

An Ethernet station able to operate at an average throughput in excess of these averages could be used effectively as controllers. Applications with a total average network throughput demand of less than the 10 megabits per second are ideally suited. Further, such applications allow fair and graceful access during peak load situations.

Related to throughput is the issue of average response time. In some environments it is critical to ensure that a command and response can be exchanged in a fixed interval. The Ethernet access method is probabilistic because of the possibility of a collision occurring. Automatically repeated access attempts reduce the probability of nondelivery to an increasingly smaller value with each attempt. The frame will be declared undeliverable if 16 consecutive access attempts fail due to collisions. Thus, the sending station is notified if the network load is inappropriate for the chosen application. The maximum time for access or notification of nondelivery is related to the maximum block sizes permitted by the network and fixed network parameters. This maximum time can be a very large time.

Other factors besides the access method play a role in the average response time. Environmental noise and the station receive window (see Section 5) must be considered.

In the presence of noise, there is a probability that some frames will be damaged and lost. Any station receiving a frame with a frame-check-sequence error discards it without further action. The client layer protocols must provide error recovery procedures with response timers and sequence verification for detecting lost blocks. The maximum response time possible before delivery or notification of failure is related to the client layer time-out value and maximum retry count.

Finally, each receiving station may accept blocks at a burst rate much higher than its ability to process such blocks. Eventually, some limited resource within the receiving station will be consumed and subsequent data cannot be received until processing of prior data is completed. The receive station recovery time (or window) determines the period where a station would miss blocks addressed to it, thereby invoking time-out recovery at higher levels of protocol. With respect to the ISO Model, the transport control layer would be responsible for such time-out recovery.

In an environment where response time is critical, the selection of Ethernet obligates the designer to consider sustained average load, peak load durations, time-out recovery interval, and receive station recovery time. The designers goal would be to avoid excessive collisions and lost blocks.

Excessive collisions are avoided by keeping the average load to a value less than the maximum capacity. Loads of 40% to 80% capacity will experience few collisions.

Lost blocks are avoided by ensuring that transmissions to a receiving station do not exceed the rate at which they can be processed. This is usually accomplished by a flow control acknowledgement packet provided by the client layer software. Further, the number of stations transmitting to a single receiving station must not exceed the number of frames that can be received back-to-back.

4 ETHERNET* IN THE OFFICE AND FACTORY

Interconnection of computers in the office has received considerable attention in the trade literature and vendors are supplying new products at an increasing rate.^{14, 5, 7, 19} Systems supporting shared data bases, electronic mail, shared peripherals, etc. are easy to analyze for performance requirements. In the office, other non-obvious considerations are as important as the final user application.

First, an office network must be immune to inadvertent network interruption. That is, the electrical failure (shorting or grounding) of a transceiver or controller must not cause failure of the network. Electrical spikes on the cable must not destroy the attached transceivers or stations.

Second, the network must be easy to configure or reconfigure. Nontechnical office workers must be able to move a station from one physical location to another. The physical location should be transparent to the user application supplied by the station.

Third, the network must provide a level of performance for expected applications and for future uses. Growth of the office and addition of higher performance stations must be considered. Even the requirements for real-time data such as voice or video must be considered.

Finally, networks selected for the office must eventually achieve widespread and multivendor support. This ensures the continued development of products and continued effort to reduce the connection costs. Certainly, an acceptable network must be accessed using single VLSI silicon chips to allow very low cost interconnect.

Ethernet fits the office requirements very well. The design is flexible and permits widely varied uses. The successful effort to standardize Ethernet will enhance the usefulness of a VLSI solution for Ethernet and make connection more cost effective.

In the factory, applications are more likely to have distributed sampling and decision making characteristics. In such environments, real-time responses, reliability and noise immunity are important factors. Ethernet can be used in the factory environment where noise is reduced to the levels stated in the Specification and load levels are sufficiently low to ensure a small probability of repeated collisions. However, concern for noise immunity of the network must be matched by similar concern for the station boards and components which are also susceptible to noise.

An Ethernet network carrying critical time-dependent data should not be carrying a full load of noncritical data since no priority scheme is provided. That is, there is no method assigning a higher priority for network access to a station with time-critical data.

Ethernet has been specified to serve the greatest number of applications while retaining a conceptually simple algorithm. Ethernet does not provide some features which have been proposed for various local area networks. It does not provide a method of allocating a guaranteed bandwidth to a specific station. Proposals to provide such a feature have been made for various alternative networks^[1, 16, 22, 23] including the IEEE 802 committee proposal. These generally involve alternative backoff algorithms or the passing of a token to signal the access of the network.

These alternatives avoid the problems of random delays due to collision resolution. The problems of noise, time out resolution, and receiving window remain.

The IEEE 802 proposal defines a deterministic system in terms of the absence of noise.^[14] The guaranteed access in the absence of noise is achieved with increased complexity of the individual station's access algorithm. Methods must be provided for initial startup or recovery from a lost token and for detection and removal of duplicate tokens. Further complexity is added by the negotiation for allocation or refusal of requests for guaranteed bandwidth.

Ethernet ensures multivendor compatibility of all Ethernet implementations by not providing optional features. This results in all stations being compatible on all Ethernet networks everywhere. The advantage of this feature is superior

to the advantages of features such as variable bit rates or variable address lengths.

5 THE ISBC® 550 ETHERNET® COMMUNICATIONS CONTROLLER

Intel's first OEM product for Ethernet provides an interface for MCS-85, iAPX 86, or iAPX 88-based microcomputer systems. The controller is a powerful and full-featured implementation of the Ethernet Specification.

The controller consists of two boards; a processor board and a serial/deserialization (SERDES) board. The processor board includes firmware for the packet management to the data link layer and communication with a system processor board. The serialization/deserialization board performs most of the functions specified in the Ethernet Specification.

The processor board contains a 5 MHz 8088 CPU and 8K bytes of firmware for the data link layer and the MULTIBUS Interprocessor Protocol (MIP)^[28] implementations. Communications between Ethernet processor board and the system processor board (iSBC 86/12A, 86/30, etc.) is accomplished via MULTIBUS memory. (See Figure 5-1).

The Ethernet data link layer is accessed and controlled by software in the host system by the External Data Link (EDL) layer. This software appears as a protocol bridge since the higher level client layer software cannot directly access the Ethernet data link layer. The processor board also provides packet encapsulation for packets to be transmitted.

Two other methods of communication are also used between the system and controller processor boards. First, the Ethernet processor board contains a jumper selectable base address of a MULTIBUS memory block where initialization information is to be exchanged. Code in the system processor must be configured to use the initialization addresses expected by the iSBC 550 Ethernet Controller. Second, a configurable controller wakeup address (I/O port address) is selectable by jumpers on the Ethernet processor board. The host processor board uses this address to interrupt the iSBC 550 Ethernet Controller when a command is available.

The Ethernet processor board executes the commands received by the External Data Link (EDL) layer. Data to be sent is moved to static RAM on the processor board from MULTIBUS memory specified by EDL commands. Received data is moved from on-board static RAM whenever MULTIBUS memory is made available by an EDL command.

The processor board contains 8K of static RAM for transmission and receipt of Ethernet frames. This memory

is divided into 4 buffers of 2K each. One buffer is allocated for transmission of a frame and three allocated for receipt of frames. These buffers are used to avoid contention with access to MULTIBUS memory from other boards in the system.

For transmission of a frame, data is moved from MULTIBUS memory specified by an EDL command to the static RAM buffer. DMA memory transfers are performed by the SERDES board during the actual transfer to the cable.

Up to 3 frames can be received before the static RAM is filled. Once filled the data in the static RAM must be moved to MULTIBUS memory specified by an EDL command. While the static RAM is full and data is being copied to MULTIBUS memory, any frames destined for this station will be lost. The small window where the controller is not accepting incoming frames is resolved by client layer flow control procedures.

The iSBC 550 processor board provides 16K of dynamic RAM which can be loaded from the system processor. This memory is provided for support of higher layers of network software such as the ISO session, transport and network layers. This feature is not currently offered for

OEM usage of the iSBC 550 Ethernet Controller but is expected to be used when Intel Development Systems are interconnected via Ethernet. Any client level software developed for the iSBC 550 Ethernet Controller should be designed with strict separation of the ISO layers, particularly between presentation and session. The user is thereby better prepared for evolution of Ethernet products.

The SERDES board performs the serialization/deserialization, framing, frame-check-sequence generation and checking, Manchester encoding and decoding, address recognition and all CSMA/CD link access protocol functions. Data is sent or received by DMA access to the static RAM on the processor board.

The Model 675 Development System for Ethernet provides the user with the tools necessary to develop and test communication software and applications which will use Ethernet as a communications method. It contains a MULTIBUS Ethernet Controller, a 10-meter cable for connecting model 675s, and an Ethernet Data Link layer software library. The library contains an implementation of the MULTIBUS Interprocessor Protocol (MIP) and EDL interface modules to simplify the user interface to Ethernet.

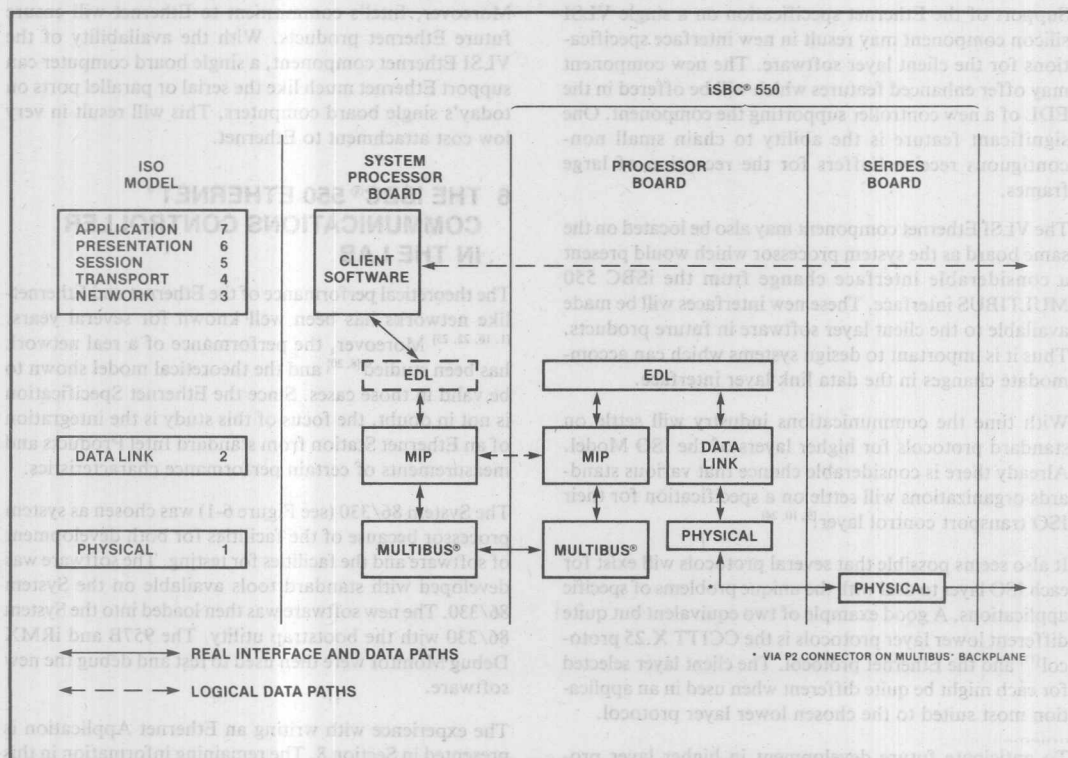


Figure 5-1. iSBC® 550 Protocol Architecture

The Model 677 DS/E Ethernet upgrade kit for Intel Development Systems provides the Series II/85 or Series III user with the additional tools necessary to develop and test software that will use Ethernet.

For target systems, the iRMX Operating Systems are ideal for a great variety of situations. The iMMX 800 MULTIBUS Message Exchange Software^[28] is the standard iRMX Operating System implementation of MIP and may be used for communication with the iSBC 550 Ethernet Controller. When the iRMX 86 Operating System is used, a standard driver, the E-Driver, is provided for the Basic I/O system. The E-Driver performs the initialization procedures for the iSBC 550 Ethernet Controller and allows the client layer to use the standard iRMX 86 I/O system interface.

The interface offered by the iRMX 86 E-Driver is not identical to that offered by the Model 675 Development System. This is expected due to the different system architectures. Designs of client layers should isolate the interface requirements for lower layers from the bulk of the software supporting the higher level protocol. This design practice eases the integration of future Ethernet developments.

Support of the Ethernet specification on a single VLSI silicon component may result in new interface specifications for the client layer software. The new component may offer enhanced features which will be offered in the EDL of a new controller supporting the component. One significant feature is the ability to chain small non-contiguous receive buffers for the reception of large frames.

The VLSI Ethernet component may also be located on the same board as the system processor which would present a considerable interface change from the iSBC 550 MULTIBUS interface. These new interfaces will be made available to the client layer software in future products. Thus it is important to design systems which can accommodate changes in the data link layer interface.

With time the communications industry will settle on standard protocols for higher layers of the ISO Model. Already there is considerable chance that various standards organizations will settle on a specification for their ISO transport control layer.^[9, 10, 24]

It also seems possible that several protocols will exist for each ISO layer to deal with the unique problems of specific applications. A good example of two equivalent but quite different lower layer protocols is the CCITT X.25 protocol^[17] and the Ethernet protocol. The client layer selected for each might be quite different when used in an application most suited to the chosen lower layer protocol.

To anticipate future development in higher layer protocols, client layer software must be designed with strict

adherence to the definitions of the ISO layers. Further, the layer boundaries must be engineered to eliminate tight dependencies between implementations of each layer.

The iSBC 550 Ethernet Controller is a powerful and full-featured implementation of the Ethernet Specification. Its intent is to provide a development tool for the verification of client layer software. With the future availability of the VLSI Ethernet component, client layer software can be ported to the target systems.

For systems which must be operational before the VLSI component is available, the iSBC 550 Ethernet Controller is fully capable of providing the Ethernet function. Standard iSBC board level products can be used to create complete systems which use Ethernet to communicate without long development times. With the System 86/330, a complete microcomputer system, development time can be reduced even more. Addition of the iMMX 800 software and the iRMX 86 E-Driver permits the engineering focus to be on the client layer software and the user application.

It is easy to get started with Ethernet using the iSBC 550 Ethernet Controller and System 86/330. Intel's commitment to providing the board level and component level products permits the user to buy or build as appropriate. Moreover, Intel's commitment to Ethernet will ensure future Ethernet products. With the availability of the VLSI Ethernet component, a single board computer can support Ethernet much like the serial or parallel ports on today's single board computers. This will result in very low cost attachment to Ethernet.

6 THE iSBC® 550 ETHERNET® COMMUNICATIONS CONTROLLER IN THE LAB

The theoretical performance of the Ethernet and Ethernet-like networks has been well known for several years.^[1, 16, 22, 25] Moreover, the performance of a real network has been studied^[8, 20] and the theoretical model shown to be valid in those cases. Since the Ethernet Specification is not in doubt, the focus of this study is the integration of an Ethernet Station from standard Intel Products and measurements of certain performance characteristics.

The System 86/330 (see Figure 6-1) was chosen as system processor because of the facilities for both development of software and the facilities for testing. The software was developed with standard tools available on the System 86/330. The new software was then loaded into the System 86/330 with the bootstrap utility. The 957B and iRMX Debug Monitor were then used to test and debug the new software.

The experience with writing an Ethernet Application is presented in Section 8. The remaining information in this section presents the results of the performance tests.

To understand the performance of the iSBC 550 Controller, three measurements were proposed:

- The rate at which a station could present data to the Ethernet was measured to determine if receive window closure could be caused by a single uncontrolled station.
- The rate of reliable data exchange was measured for analysis of the controller performance in a resource server.
- The round trip response time for reliable data exchange was measured for analysis of controller performance in real-time control environments.

A test program was written to obtain the three measurements. In addition the program was to run as an iRMX 86 task to measure the performance of standard products and to run as a custom system. This objective was achieved by structuring the test program into layers.

The test program consisted of three modes: output only, initiator and responder. In each case, the data in each out-

put buffer was preset, and thus does not include application processing overhead. The program would periodically request statistics from the iSBC 550 Ethernet Controller and create a display of various statistics.

In output only mode the station would transmit data into the network at the maximum rate possible. Any received input data is discarded. This test was to determine if the receive window, as discussed earlier, could be closed by a single station. In fact, when a second station was set up to receive this continuous output, nearly half the blocks transmitted were lost due to window closure.

The initiator and responder were created to measure the rate of reliable data transfer which could be achieved without causing window closure. The initiator will send a packet with a specific sequence number. Upon receipt of a response with the expected sequence number the initiator would send another packet with the next available sequence number; modulo 2^{16} . The initiator can transmit more than one block in sequence before requiring acknowledgement. The number of blocks outstanding could be varied for the test.

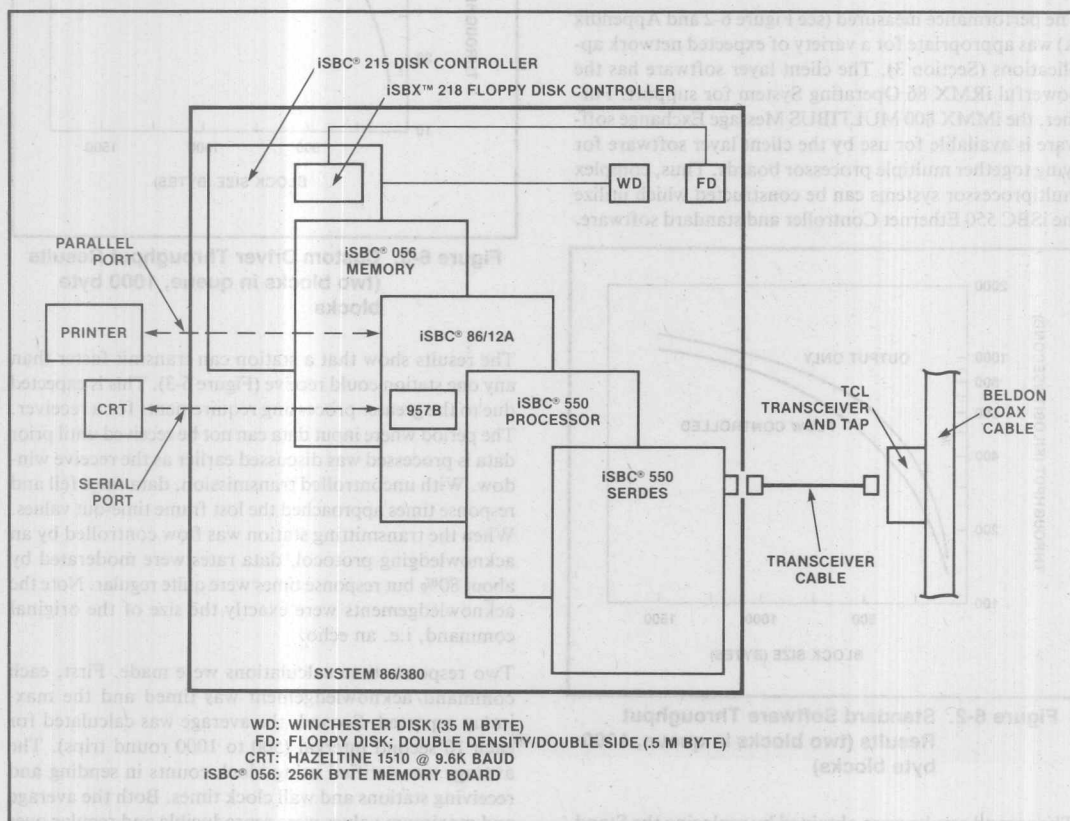


Figure 6-1. Ethernet* Test System

Whenever a response is received out of order, the response is discarded. Either the original block or the response was lost for the expected sequence number. No special reject logic was implemented. Lost blocks are recovered by retransmission after an acknowledgement timeout.

The responder would always echo each received data block with a response of exactly the same size. The response carries the same sequence number as the original block. No check is made to ensure the block is received in order nor is a reject response sent for out-of-sequence errors. Such a check would be required if the received data must be processed in correct sequence.

The first results measured the performance obtainable with standard software and hardware:

- System 86/330
- iRMX 86 Operating System, Release 4
- iMMX 800 MULTIBUS Message Exchange, Release 2
- E-Driver (part of iMMX 800 MULTIBUS Message Exchange)

The performance measured (see Figure 6-2 and Appendix A) was appropriate for a variety of expected network applications (Section 3). The client layer software has the powerful iRMX 86 Operating System for support. Further, the iMMX 800 MULTIBUS Message Exchange software is available for use by the client layer software for tying together multiple processor boards. Thus, complex multiprocessor systems can be constructed which utilize the iSBC 550 Ethernet Controller and standard software.

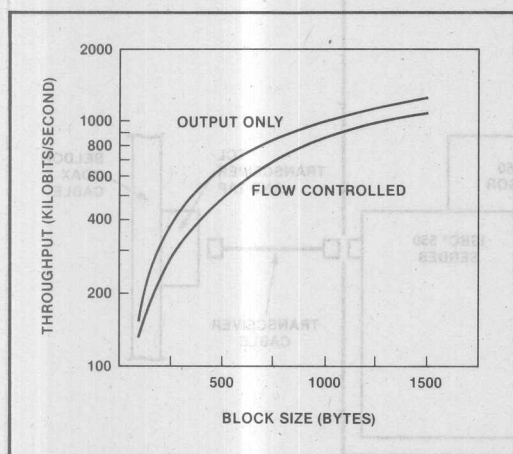


Figure 6-2. Standard Software Throughput Results (two blocks in queue, 1000 byte blocks)

The second results were obtained by replacing the Standard Operating System software with a custom driver for

the iSBC 550 Ethernet Controller. This driver consists of a version of the MULTIBUS Interprocessor Protocol (MIP) as specified in the iSBC 550 Programmers Reference Manual. In this version the test programs did not use any iRMX 86 System calls.

The test results show that reliable data rates of up to 1 megabit per second can be achieved using the iSBC 550 Ethernet Controller (Figures 6-3 and Appendix A). This is easily fast enough to support a wide variety of applications.

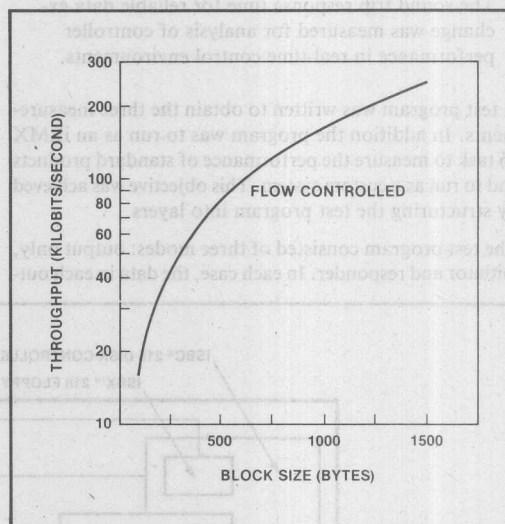


Figure 6-3. Custom Driver Throughput Results (two blocks in queue, 1000 byte blocks)

The results show that a station can transmit faster than any one station could receive (Figure 6-3). This is expected due to the greater processing requirements for a receiver. The period where input data can not be received until prior data is processed was discussed earlier as the receive window. With uncontrolled transmission, data rates fell and response times approached the lost frame time-out values. When the transmitting station was flow controlled by an acknowledging protocol, data rates were moderated by about 80% but response times were quite regular. Note the acknowledgements were exactly the size of the original command, i.e. an echo.

Two response time calculations were made. First, each command/acknowledgement was timed and the maximum reported. Second, the average was calculated for each 10 second interval (500 to 1000 round trips). The average was verified using block counts in sending and receiving stations and wall clock times. Both the average and maximum values were reproducible and regular over long periods of time (20 minutes).

The maximum response time represents the worst case round trip time for a message originating in one application, echoed by another application in a remote System 86/330, and returned to the original application. This number is equivalent to the average response time in tests where only one block is allowed in the network.

When two blocks are allowed in the network, the average time improves because overlapped processing can occur. That is, data blocks can be prepared and sent while prior data blocks are echoed in the remote system. When more than two blocks are allowed in the network, some get queued either locally or remotely, and this results in increased maximum response times. (See Figure 6-4 and Appendix A). Queueing occurs, either locally or remotely, when a frame has been given to the Ethernet controller but has not yet been sent. It also occurs when a frame has been received by the controller but the frame has not been received by the application.

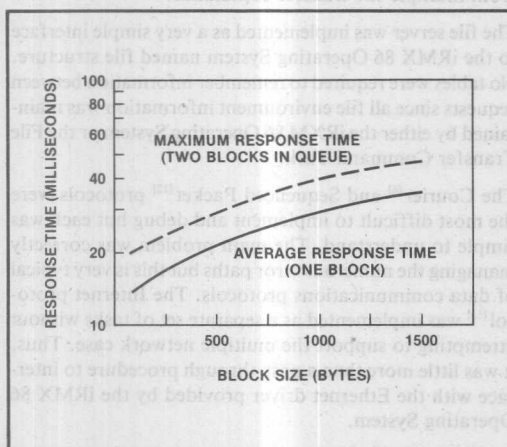


Figure 6-4. Custom Driver Response Time Results (1000 byte blocks)

As a result of the measurements of the Ethernet controller in the System 86/330, additional tests were conducted by the Data Communications Software Development Group. These measurements were conducted with software probes to determine the time to perform various operations. The results of these tests are reported in Appendix B. A custom Ethernet data link interface was developed to use standard iRMX 86 interrupt and mailbox facilities. These results show a very high performance level achievable with the full availability of services of the iRMX 86 Operating System.

The programs used to obtain these results can be obtained from the Insite Library. The programs are included with the application discussed in Section 8 and titled "Xerox File Transfer Facility."

7 A TEST IN AN ELECTRICALLY NOISY ENVIRONMENT

The ability of Ethernet to operate in a factory environment is sometimes questioned. A factory environment is usually described as one where various levels of electrical noise are expected. Since no actual tests have been published, we took our Ethernet test systems into the main equipment rooms where we had access to an arc welder, a large induction motor and several motors and generators.

Two simple tests were performed to evaluate the verbal criticism of Ethernet. These simple tests were only intended to demonstrate the use of Ethernet in one environment which contained unspecified levels of electrical noise. Additional studies are needed to draw any conclusions from these tests since the results are not reproducible anywhere else but in our test site.

The first test was to attempt to cause a failure of the Ethernet in an environment containing motors and generators. The second test was to cause a failure by actively welding near the cables and transceivers.

Our first test was to determine the error rate in our own development lab. The test system consisted of two System 86/330s, two iSBC 550 Ethernet Controllers, 150 feet of coaxial cable, two 50 foot transceiver cables, and two transceivers. The response time test discussed earlier was run for 13.5 hours without detecting a collision or a frame-check-sequence error.

This same equipment was moved to the equipment room of our facility where there were several motors and generators. The coaxial cable, transceivers and transceiver cables were placed in the equipment room at close proximity to expected noise sources. No special effort was made to protect the cable or other devices from noise.

The results in the equipment room surprised us after the experience in the development lab. The error rate was one damaged block for each 10,000 blocks transmitted (see Appendix A).

One previously unknown characteristic of Ethernet is the immediate detection of frames damaged by noise by the collision detect logic. Only 12 frames were lost during the 3 hour test. These lost frames were all recovered by the client layer response time-out recovery logic. Most of these errors could have been recovered by a packet reject logic as found in Intel's Network Architecture transport layer protocol.

The noise test was rerun with the transceivers and transceiver cables outside the equipment room. The error rate dropped somewhat (see Appendix A). The transceivers and transceiver cables are not the sole source of noise induction. In harsh environments where noise immunity

specifications are required the cables can be protected by additional shielding such as conduit.

To test Ethernet near an arc welder, we laid the coaxial cable around the welding table and placed both transceivers within 10 feet of the welder. The coaxial cable was within 3 feet of the arc and within 6 inches of the high voltage leads. This physical configuration was arbitrary.

The first test failed when the operator inadvertently allowed the high voltage cables to touch the coaxial cable. Inadequate grounding of the coaxial cable lead to the failure of both CPU boards. However, both systems were immediately reloaded and no damage was done to any component.

A moderate error rate was present on the cable but there was no evidence that welding caused any induced noise in this configuration. Both AC and DC welding were tried, each for about 5 minutes. In each case the currents were so high that the arc was too hot for most welding needs. In any case, we could not disrupt the network with the arc welder under these conditions.

These simple tests demonstrate that at least one configuration can be successfully supported by Ethernet in a noisy environment. Additional testing would be required to draw any conclusions about any other general situation.

8 AN ETHERNET* APPLICATION

The selection and use of a local area network must depend on the user application. One of the common concerns of customers is the amount of effort to implement various network protocols and common network applications.

To understand this concern, we chose to implement the Xerox protocols^[6,12] since they demonstrate multivendor interconnectability. These protocols are only one of several choices which are available to network developers. The Intel network products^[19] offer protocols which allow access to file servers, print servers and development systems. Eventually, Intel will offer data base managers and other commercial applications.

Several vendors have selected the Department of Defense Protocol,^[9,10] which is compatible with Arpanet and is becoming an industrial standard. The European Computer Manufacturers Association (ECMA) Transport Protocol is another choice which will be effective in Europe. Proprietary network protocols of mainframe and mini-computer manufacturers may also be attractive to some network developers as may special purpose, one-of-a-kind, protocols.

The Xerox protocols were chosen because they could be used to demonstrate a multivendor interconnect using Ethernet. They also were chosen since they were based on an operational system which reduced the risk of logic flaws. Further, the protocols were conceptually easy to

understand and structured into layers. Finally, only portions of the protocol needed to be implemented to demonstrate a multivendor file transfer facility.

We chose to implement a file transfer facility and a primitive file server on the System 86/330. The file transfer facility was implemented as a non-resident command of the iRMX 86 Operating System. The command offered a user interface similar to the standard Human Interface COPY command. The file transfer command was very easy to implement since the iRMX 86 Human Interface offers standard system calls for most of the features needed.

The file server was implemented as a permanent task which could OPEN, CLOSE, READ or WRITE a file. The file server protocol was implemented from the example in the Xerox Courier publication^[6] since the file server protocol was not available from Xerox at the time of implementation. The file server can process requests from multiple file transfer commands.

The file server was implemented as a very simple interface to the iRMX 86 Operating System named file structure. No tables were required to remember information between requests since all file environment information was maintained by either the iRMX 86 Operating System or the File Transfer Command Task.

The Courier^[6] and Sequenced Packet^[12] protocols were the most difficult to implement and debug but each was simple to understand. The main problem was correctly managing the numerous error paths but this is very typical of data communications protocols. The Internet protocol^[12] was implemented as a separate set of tasks without attempting to support the multiple network case. Thus, it was little more than a pass-through procedure to interface with the Ethernet driver provided by the iRMX 86 Operating System.

Each layer of the Xerox protocol corresponds to a layer of the ISO Model.^[15] Note that the session layer is not required nor provided in the Xerox architecture (see Figure 8-1). Each layer of protocol was implemented as a separate set of tasks, each communicating with the adjacent layer via operating system message exchanges. In some cases, the layers were implemented as several dependent tasks to ease the problem of processing unsolicited requests for service from both adjacent layers. It also allows the existence of alternate implementations of an adjacent layer protocol, but this feature was not explored.

The code consisted of 4000 lines of PLM statements and occupied 12K bytes of memory (see Figure 8-2). Additional memory was needed to support data buffering and various system table structures. The dynamic memory requirements were about 10K in our example. The memory requirement for both code and buffering can be reduced by using the compilers optimization options and through careful memory resource management.

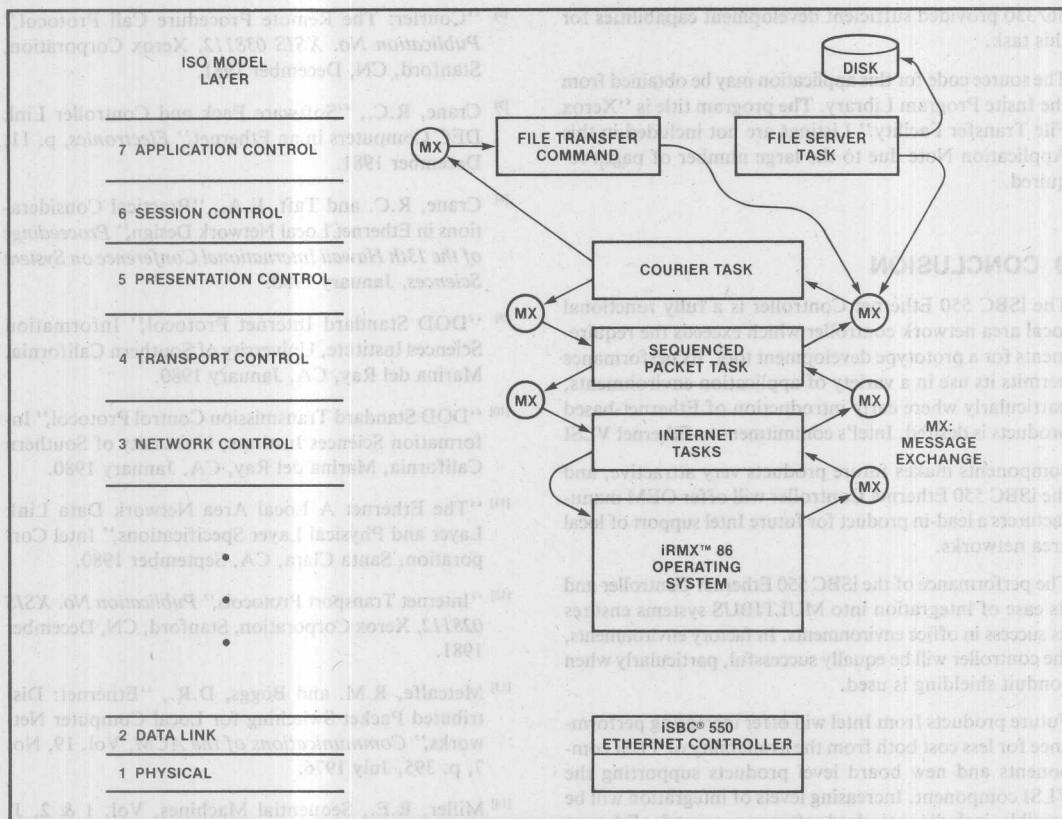


Figure 8-1. Application Module Structure

Module	Lines of Code	Memory (code only)
1. Internet Protocol Tasks	675	1.8K
2. Sequence Packet Tasks	690	2.8K
3. Courier Tasks	1100	4.0K
4. File Server Task	570	1.4K
5. File Transfer Command	760	1.9K
	3375	11.9K

Figure 8-2. Application Memory Requirements

This code represents about 2 months of work to study, implement and unit test. However, the development of this code as a standard product with adequate documentation and system test would require additional work. Using equations available in the Mythical Man Month^[1] this code would require 12 months effort. The inclusion of enhanced features of the network could easily increase this implementation multifold. These enhanced features would

include network statistics, network gateways, internet datagrams, etc.

The Ethernet Application was very easy to debug due to the iRMX debug monitor offered by System 86/330. A moderate speed printer was essential to this task since hard copy listings and memory maps were necessary. The system performed very well as a development station, partly due to the high performance of the Winchester Disk.

The developers of applications for Ethernet can choose one of two ways to implement higher level code. First, the Model 675 Development Station for Ethernet can be chosen which brings an Ethernet controller and all the development tools available to microprocessor development. (One can also add the Model 667 DS/E upgrade to an existing Intellec Series II or III.) This option allows the use of In-Circuit Emulators and the support of EPROM programmers. A second choice is the iSBC 550 Ethernet Controller which is added to a MULTIBUS system. The latter was chosen for this Application Note since the System

86/330 provided sufficient development capabilities for this task.

The source code for this application may be obtained from the Insite Program Library. The program title is "Xerox File Transfer Facility." Listings are not included in this Application Note due to the large number of pages required.

9 CONCLUSION

The iSBC 550 Ethernet Controller is a fully functional local area network controller which exceeds the requirements for a prototype development tool. Its performance permits its use in a variety of application environments, particularly where early introduction of Ethernet-based products is desired. Intel's commitment to Ethernet VLSI

components makes future products very attractive, and the iSBC 550 Ethernet Controller will offer OEM manufacturers a lead-in product for future Intel support of local area networks.

The performance of the iSBC 550 Ethernet Controller and its ease of integration into MULTIBUS systems ensures its success in office environments. In factory environments, the controller will be equally successful, particularly when conduit shielding is used.

Future products from Intel will offer increasing performance for less cost both from the availability of VLSI components and new board level products supporting the VLSI component. Increasing levels of integration will be possible, including standard software support for Ethernet Interfaces both for Intel Development Systems and iRMX 86 Operating Systems.

10 REFERENCES

- [1] Ames, G.T. and Lazowski, E.D., "Behavior of Ethernet-like Computer Communications Networks," *Technical Report No. 79-05-01*, Dept. of Computer Science, University of Washington, April 1979.
- [2] Boggs, D.R., et al, "Pup: An Internetwork Architecture," *IEEE Transactions on Communications*, Vol. COM-28, No. 4, p. 612, April 1980.
- [3] Brooks, F.P., *The Mythical Man-Month*. Reading, MA: Addison Wesley, 1972, pg. 90-92.
- [4] Clark, D.D., et al, "An Introduction to Local Area Networks," *Proceedings of the IEEE*, Vol. 6, No. 11, November 1978.
- [5] Cotton, I.W., "Technologies for Local Area Computer Networks," *Proceedings of the LACN Symposium*, May 1979.
- [6] "Courier: The Remote Procedure Call Protocol," *Publication No. XSYS 038112*, Xerox Corporation, Stanford, CN, December 1981.
- [7] Crane, R.C., "Software Pack and Controller Link DEC Computers in an Ethernet," *Electronics*, p. 11, December 1981.
- [8] Crane, R.C. and Taft, E.A., "Practical Considerations in Ethernet Local Network Design," *Proceedings of the 13th Hawaii International Conference on System Sciences*, January 1980.
- [9] "DOD Standard Internet Protocol," Information Sciences Institute, University of Southern California, Marina del Ray, CA, January 1980.
- [10] "DOD Standard Transmission Control Protocol," Information Sciences Institute, University of Southern California, Marina del Ray, CA, January 1980.
- [11] "The Ethernet A Local Area Network Data Link Layer and Physical Layer Specifications," Intel Corporation, Santa Clara, CA, September 1980.
- [12] "Internet Transport Protocols," *Publication No. XSYS 028112*, Xerox Corporation, Stanford, CN, December 1981.
- [13] Metcalfe, R.M. and Boggs, D.R., "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, Vol. 19, No. 7, p. 395, July 1976.
- [14] Miller, R.E., *Sequential Machines*, Vol. 1 & 2, J. Wiley.
- [15] "Open Systems Interconnection," International Organization for Standardization, ISO/TC 97/SC 16, May 1980.
- [16] Park, C. and Lynch, J., "Relative Performance of Tokens vs. CSMA/CD," *Letter to Members of the IEEE DLMAC Subcommittee*, Hewlett Packard, Cupertino, CA, November 1980.
- [17] "Recommendation X.25," International Telegraph and Telephone Consultative Committee (CCITT), AP V11-No 7-E, Geneva, June 1980.
- [18] Riley, W.B., "LAN Standards Controversy Looms: Ethernet vs. IEEE-802," *Electronic Design*, p. SS-25, September 1981.
- [19] Ryan, R., Marshall, G., Beach, R., and Kerman, S., "Local Network Architecture Proposed for Work Stations," *Electronics*, p. 120, August 1981.
- [20] Shoch, J.F. and Hupp, J.A., "Measured Performance of an Ethernet Local Network," *Communications of the ACM*, Vol. 23, No. 12, p. 711, December 1980.

- [21] Shoch, J.F., et al., "Evolution of the Ethernet Local Computer Network," to appear in the IEEE Computer Magazine, 1982.
- [22] Spaniol, O., "Modeling of Local Computer Networks," *Computer Networks*, p. 315, 1979.
- [23] Spector, A.Z., "Performing Remote Operations Efficiently on a Local Computer Network," *Report No. STAN-CS-80-850*, Department of Computer Sciences, Stanford University, December 1980.
- [24] "Standard ECMA-72 Transport Protocol," European Computer Manufacturers Association, January 1981.
- [25] Tobagi, F.A., "Message-based Priority Functions in Multiaccess/Broadcast Communication Systems with a Carrier Sense Capability," *Proceedings of the Local Area Communications Network Symposium*, Boston, MA, May 1979.
- [26] Zimmerman, H., "OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications*, Vol. COM-28, No. 4, April 1980.

11 INTEL RELATED PUBLICATIONS

- [27] Ethernet Communications Controller Programmer's Reference Manual (121769-001)
- [28] iMMX 800 Software Reference Manual and Users Guide (143808-002)
- [29] Introduction to the iRMX 86 Operating System (9803124-02)
- [30] iSBC 550 Ethernet Communications Controller Hardware Reference Manual (121746-001)
- [31] System 86/330 Overview Manual (143898-001)

APPENDIX A
LAB TEST RESULTS

- 100 Zimmerman, H., "OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection," IEEE Transactions on Communications, Vol. COM-38, No. 4, April 1990.
- 101 "Intel Related Publications"
- 102 Ethernet Communications Controller Programmer's Reference Manual (111769-001)
- 103 iMMX 800 Software Reference Manual and User's Guide (143808-003)
- 104 Introduction to the iRMX 86 Operating System (0803124-03)
- 105 iSBX 550 Ethernet Communications Controller Hardware Reference Manual (111746-001)
- 106 iSystem 86/330 Overview Manual (143892-001)
- 107 Shoch, J.F., et al., "Evolution of the Ethernet Local Computer Network," to appear in the IEEE Computer Magazine, 1983.
- 108 Spaniol, O., "Modeling of Local Computer Networks," Computer Networks, p. 315, 1979.
- 109 Spitzer, A.Z., "Performing Remote Operations Efficiently on a Local Computer Network," Report No. STAN-CS-80-846, Department of Computer Science, Stanford University, December 1980.
- 110 "Standard ECMA-12 Transport Protocol," European Computer Manufacturers Association, January 1981.
- 111 Tobagi, F.A., "Message-based Priority Functions in Multicasting/Broadcast Communication Systems with a Carrier Sense Capability," Proceedings of the Local Area Communications Network Symposium, Boston, MA, May 1979.

APPENDIX A LAB TEST RESULTS

iSBC® 550 ETHERNET* CONTROLLER

Standard Software Test Results

The standard software test results were obtained with the system configuration as defined in Section 6. The following application parameters are appropriate:

block size	the size of the user data contained in an Ethernet block
block	the actual count of blocks which made the round trip in the sample interval of 10 seconds
input and output	the effective bit rate of user data in each direction
average response time	the total time interval divided by the number of blocks
maximum response time	worst case time for a block to make one round trip (10 msec clock resolution)
number of blocks in queue	the number of blocks which could be transmitted but not yet acknowledged

I. TWO BLOCKS IN QUEUE

Block Size (Bytes)	Input Blocks	Input (Kbits/sec)	Output (Kbits/sec)	Average Response Time (msec)	Maximum Response Time (msec)
100	107	8	8	93	260
200	108	17	17	93	250
500	103	41	40	98	240
1000	101	80	80	100	260
1500	103	121	121	98	280

II. ONE BLOCK IN QUEUE

Block Size (Bytes)	Input Blocks	Input (Kbits/sec)	Output (Kbits/sec)	Average Response Time (msec)	Maximum Response Time (msec)
1000	88	70	69	115	150

III. FOUR BLOCKS IN QUEUE

Block Size (Bytes)	Input Blocks	Input (Kbits/sec)	Output (Kbits/sec)	Average Response Time (msec)	Maximum Response Time (msec)
1000	101	80	80	100	470

iSBC® 550 ETHERNET* CONTROLLER

Custom Software Test Results

The custom driver results were obtained by substituting the standard iRMX 86 driver with a stand-alone driver as discussed in Section 6. The application parameters are identical for these cases as for the standard software. Case V differs from all other cases since it does not impose an end-to-end protocol. It simply presents data to the iSBC 550 controller as fast as it will transmit the data.

I. ONE OUTPUT BLOCK IN QUEUE

Block Size (Bytes)	Input Blocks	Input (Kbits/sec)	Output (Kbits/sec)	Average Response Time (msec)	Maximum Response Time (msec)
100	673	54	53	14	20
200	611	97	97	16	20
500	476	191	190	21	30
1000	349	279	279	28	30
1500	276	337	330	36	40

II. TWO OUTPUT BLOCKS IN QUEUE

Block Size (Bytes)	Input Blocks	Input (Kbits/sec)	Output (Kbits/sec)	Average Response Time (msec)	Maximum Response Time (msec)
100	792	64	63	12	30
200	702	112	112	14	30
500	585	233	234	17	40
1000	572	458	457	17	40
1500	451	546	546	22	50

III. THREE OUTPUT BLOCKS IN QUEUE

Block Size (Bytes)	Input Blocks	Input (Kbits/sec)	Output (Kbits/sec)	Average Response Time (msec)	Maximum Response Time (msec)
100	941	75	74	11	40
200	870	140	140	11	40
500	733	292	293	13	50
1000	574	459	459	17	60
1500	436	523	522	23	70

IV. FOUR OUTPUT BLOCKS IN QUEUE

Block Size (Bytes)	Input Blocks	Input (Kbits/sec)	Output (Kbits/sec)	Average Response Time (msec)	Maximum Response Time (msec)
100	847	71	71	11	50
1000	572	458	457	17	80

V. TWO BLOCKS IN QUEUE (OUTPUT ONLY)

Block Size (Bytes)	Input Blocks	Input (Kbits/sec)	Output (Kbits/sec)	Average Response Time (msec)	Maximum Response Time (msec)
100	191	0	153	—	—
200	181	0	289	—	—
500	161	0	644	—	—
1000	125	0	1008	—	—
1500	103	0	1236	—	—

ISBC® 550 ETHERNET* CONTROLLER

Noise Immunity Testing

The noise immunity tests are based on the custom software tests. The tests were all run with a block size of 1000 bytes and two blocks in the queue. The environment for these tests is described in Section 7.

The following definitions apply to these measurements:

Primary Collision	An attempt to transmit a block was prematurely ended due to the collision detection logic reporting an error. These appear to all be due to noise rather than by simultaneous access of two stations
Secondary Collision	A secondary collision is reported if a block experiences a collision upon its second or subsequent retry following a primary collision
Error Collision	An error collision is reported if 16 attempts to transport a single block are all terminated by a collision
CRC Error	A receiving station will verify the CRC of each incoming block. If incorrect the block is discarded and the error count incremented
Framing Error	A block of more than 1500 bytes is discarded as a framing error

Time-out If a block is discarded due to a CRC error or error collision, the data must be retransmitted by higher additional layers of protocol. In this case, a timer expires when a response is not received and the original block is repeated

I. COAXIAL CABLE NOISE IMMUNITY (2 hours 50 minutes)

Primary Collisions	102
Secondary Collisions	26
Error Collisions	0
CRC Errors	44
Framing Errors	0
Time-out	6

II. TRANSCEIVER/TRANSCEIVER CABLE NOISE IMMUNITY (2 hours 54 minutes)

Primary Collisions	122
Secondary Collisions	45
Error Collisions	0
CRC Errors	63
Framing Errors	0
Time-out	12

III. IMMUNITY IN VICINITY OF ARC WELDER, 200 AMP-DC (5 minutes)

Primary Collisions	2
Secondary Collisions	1
Error Collisions	0
CRC Errors	0
Framing Errors	0
Time-out	0

IV. IMMUNITY IN VICINITY OF ARC WELDER, 180 AMP-AC (5 minutes)

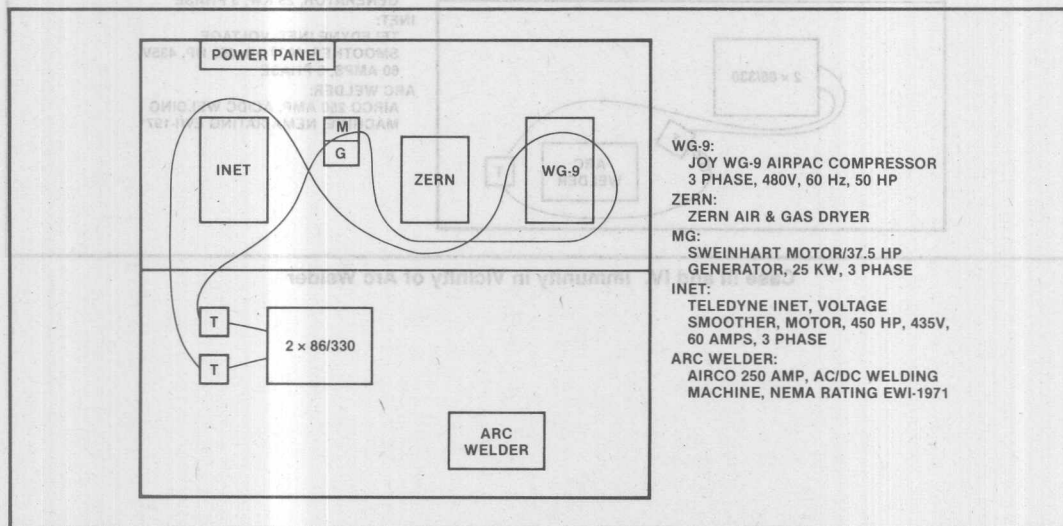
Primary Collisions	2
Secondary Collisions	1
Error Collisions	0
CRC Errors	0
Framing Errors	0
Time-out	0

V. IMMUNITY IN VICINITY OF ARC WELDER, NOT WELDING (5 minutes)

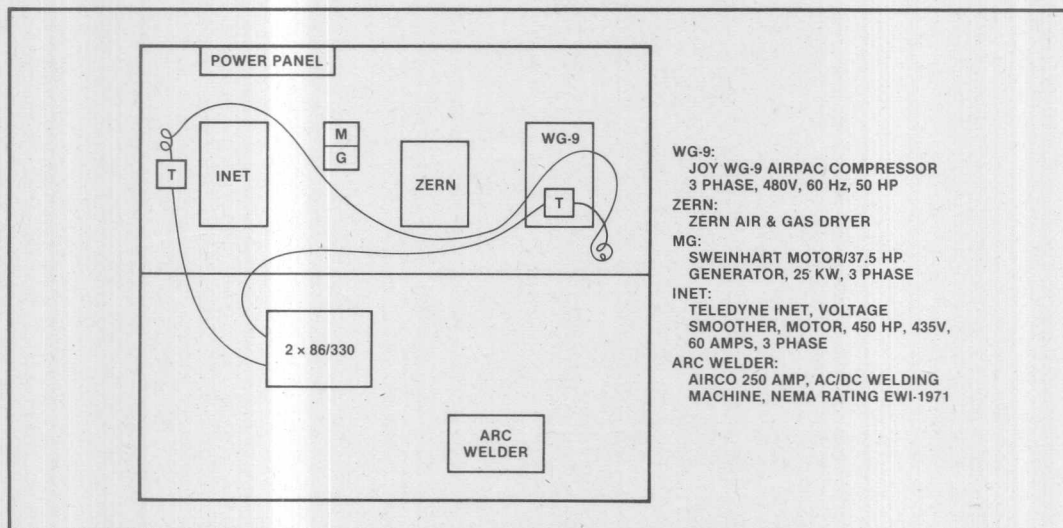
Primary Collisions	3
Secondary Collisions	2
Error Collisions	0
CRC Errors	0
Framing Errors	0
Time-out	0

VI. CASE II REPEATED IN DEVELOPMENT LABORATORY (13 hours 30 minutes)

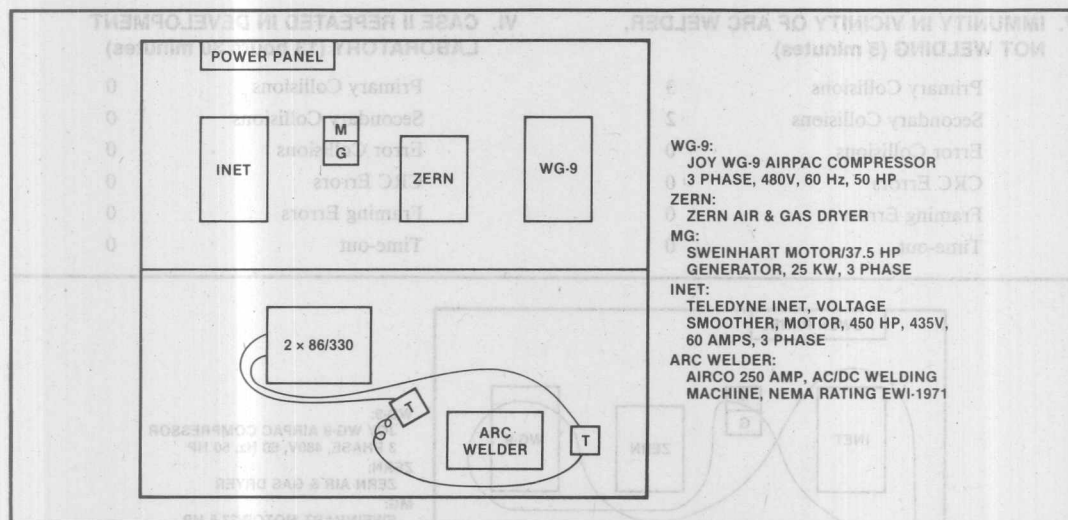
Primary Collisions	0
Secondary Collisions	0
Error Collisions	0
CRC Errors	0
Framing Errors	0
Time-out	0



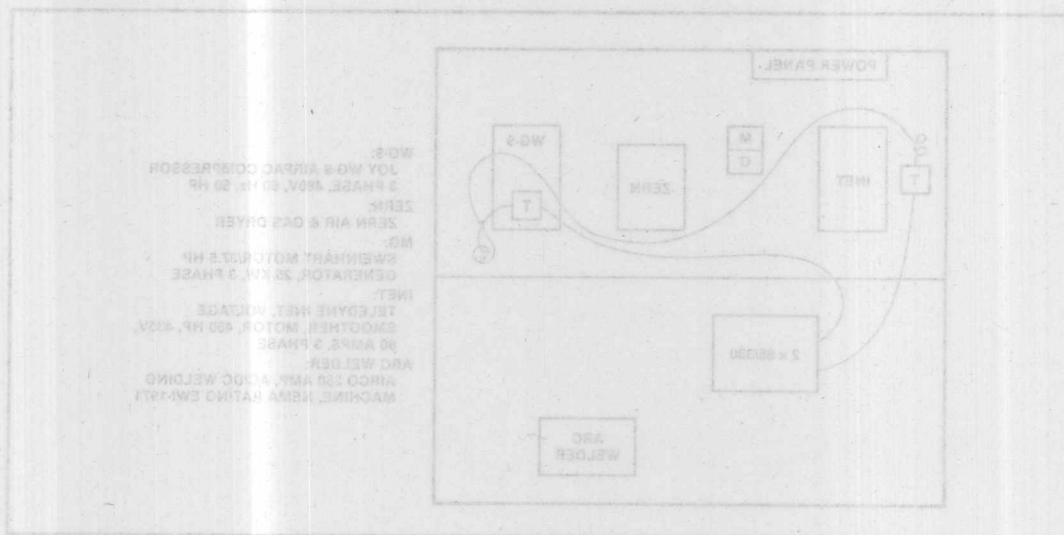
Case I. Coax Cable Noise Immunity



Case II. Transceiver/Transceiver Cable Immunity



Case III and IV. Immunity in Vicinity of Arc Welder



Case II. Transceiver/Transceiver Cable Immunity

System Test	Time to Quit One Block from User Application
Standard RMX 86 Operating System and RMX 800 MULTIBUS Exchange	45.5 milliseconds
Custom RMX 86 driver developed for this application	7.4 milliseconds
Custom RMX 86 Operating System Ethernet data link interface	19.8 milliseconds

APPENDIX B SOFTWARE PERFORMANCE RESULTS

<u>System Tested</u>	<u>Time to Output One Block from User Application</u>
Standard iRMX 86 Operating System and iMMX 800 MULTIBUS Exchange	42.5 milliseconds
Custom 86/330 driver developed for this application	7.4 milliseconds
Custom iRMX 86 Operating System Ethernet data link interface	10.8 milliseconds

APPENDIX B SOFTWARE PERFORMANCE RESULTS

AFN-01931A

Dual-port RAM hikes throughput in input/output controller board

On-board random-access memory, accessible from system bus, makes input/output controller subsystem look like just another memory board to the host microprocessor

by Craig Kinnie and Michael Maerz, Intel Corp., Santa Clara, Calif.

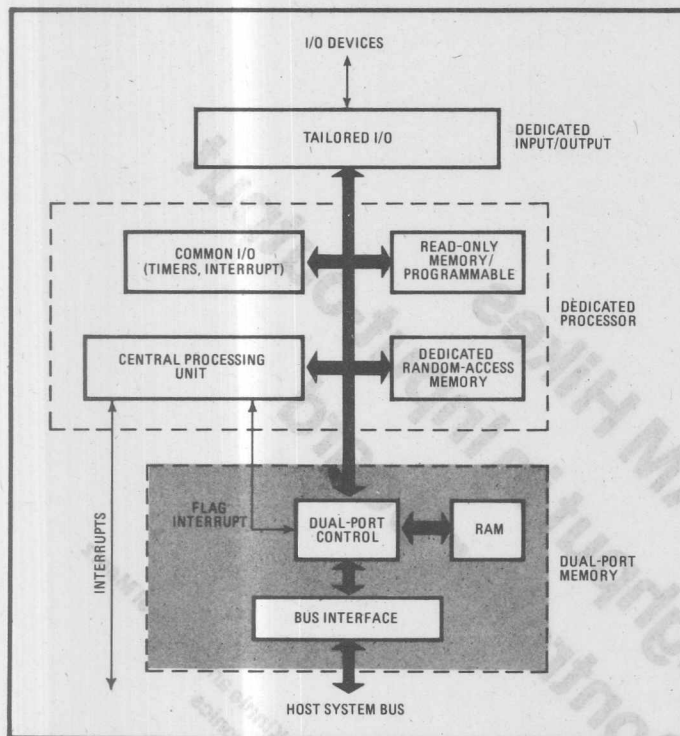
□ Input/output controllers based on microprocessors step up throughput in microcomputer systems by relieving the host processor of tedious, time-consuming control tasks—and a new design concept that increases the processing capability of this subsystem promises to hike throughput even more. It will cut the host intervention needed to transfer data and to run the controller.

In this configuration, all communications between the host processor and the controller are handled through a section of dual-port memory that resides in the controller subsystem. This setup allows more efficient transfer of large blocks of data from the I/O device to the system without contention over access to the system bus. It also

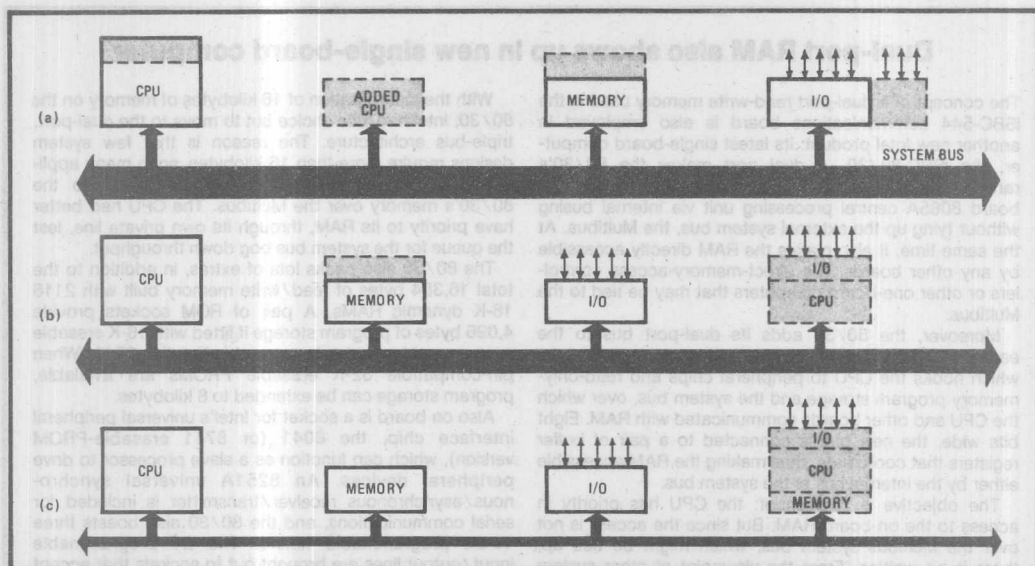
simplifies interprocessor communications because the subsystem controller appears to the host processor simply as an additional RAM board.

Although this concept allows the subsystem to remain dedicated to its I/O control function and to assume a subservient role to the host processor, it has more processing power than previous generations of such controllers. Hence it has been dubbed the intelligent-slave concept by Intel, which applies it in the iSBC 544 intelligent communications controller.

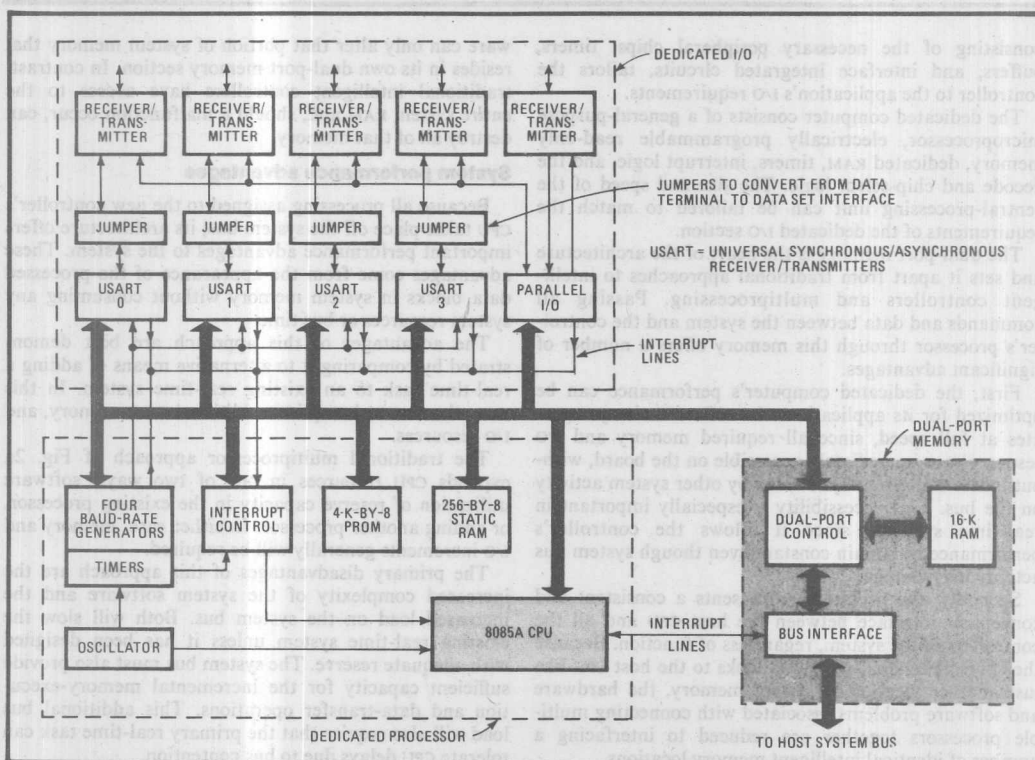
The new subsystem architecture is divided into three major sections: dedicated I/O, dedicated computer, and dual-port memory (Fig. 1). The dedicated input/output,



1. Heart of memory. New controller architecture includes the dedicated input/output circuitry and dedicated processor of an intelligent peripheral controller, but its heart is the dual-port random-access memory.



2. Performance advantages. In adding a real-time task to an existing real-time system, the load on the system bus is significantly reduced over the traditional multitasking approach (a) or the intelligent controller approach (b) by the intelligent-slave controller approach (c).



3. The 544. Based on the 8085A microprocessor with 4 kilobytes of PROM and 16 kilobytes of RAM, the subsystem is designed as a communications controller with four synchronous/asynchronous buffered serial I/O channels, and a 10-bit parallel I/O interface.

Dual-port RAM also shows up in new single-board computer

The concept of a dual-port read-write memory used in the iSBC-544 communications board is also employed in another new Intel product: its latest single-board computer, the iSBC-80/30. A dual port makes the 80/30's random-access memory directly accessible by the on-board 8085A central processing unit via internal busing without tying up the external system bus, the Multibus. At the same time, it also makes the RAM directly accessible by any other boards, like direct-memory-access controllers or other one-board computers that may be tied to the Multibus.

Moreover, the 80/30 adds its dual-port bus to the earlier iSBC computers' pair of buses: an internal bus, which hooks the CPU to peripheral chips and read-only-memory program storage and the system bus, over which the CPU and other boards communicated with RAM. Eight bits wide, the new bus is connected to a pair of buffer registers that coordinate, thus making the RAM accessible either by the internal bus or the system bus.

The objective is throughput: the CPU has priority in access to the on-board RAM. But since the access is not over the Multibus system bus, which might be tied up, there is no waiting. From the viewpoint of other system boards, the system bus is accessible a greater percentage of the time.

With the incorporation of 16 kilobytes of memory on the 80/30, Intel had little choice but to move to the dual-port, triple-bus architecture. The reason is that few system designs require more than 16 kilobytes, so in many applications all boards will be demanding access to the 80/30's memory over the Multibus. The CPU had better have priority to its RAM, through its own private line, lest the queue for the system bus bog down throughput.

The 80/30 also packs lots of extras, in addition to the total 16,384 bytes of read/write memory built with 2116 16-K dynamic RAMs. A pair of ROM sockets provide 4,096 bytes of program storage if fitted with 16-K erasable programmable read-only memories like the 2716. When pin-compatible 32-K erasable PROMs are available, program storage can be extended to 8 kilobytes.

Also on board is a socket for Intel's universal peripheral interface chip, the 8041 (or 8741 erasable-PROM version), which can function as a slave processor to drive peripheral devices. An 8251A universal synchronous/asynchronous receiver/transmitter is included for serial communications, and the 80/30 also boasts three 16-bit programmable timers. The 24 programmable input/output lines are brought out to sockets that accept quad line-drivers or -terminators for interfacing.

Ray Capece

consisting of the necessary peripheral chips, timers, buffers, and interface integrated circuits, tailors the controller to the application's I/O requirements.

The dedicated computer consists of a general-purpose microprocessor, electrically programmable read-only memory, dedicated RAM, timers, interrupt logic, and the decode and chip-select logic. The size and speed of the central-processing unit can be tailored to match the requirements of the dedicated I/O section.

The dual-port memory is the heart of the architecture and sets it apart from traditional approaches to intelligent controllers and multiprocessing. Passing all commands and data between the system and the controller's processor through this memory offers a number of significant advantages.

First, the dedicated computer's performance can be optimized for its applications. Its software always operates at full speed, since all required memory and I/O resources are immediately accessible on the board, without indeterminate delays caused by other system activity on the bus. This accessibility is especially important in real-time systems, since it allows the controller's performance to remain constant even though system bus activity may change.

Secondly, the architecture presents a consistent and convenient interface between the host CPU and all the controllers in the system, regardless of function. Because the controllers' dual-port RAM looks to the host CPU like just another location in system memory, the hardware and software problems associated with connecting multiple processors together are reduced to interfacing a number of identical intelligent memory locations.

Also, the architecture offers a degree of protection for the data in memory. The subsystem computer and soft-

ware can only alter that portion of system memory that resides in its own dual-port memory section. In contrast, traditional intelligent controllers have access to the entire system RAM and, should a malfunction occur, can destroy all of that memory.

System performance advantages

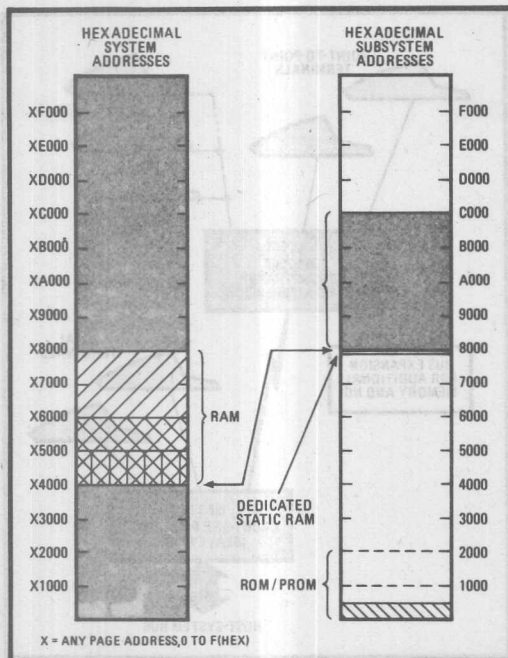
Because all processing assigned to the new controller's CPU takes place off the system bus, its architecture offers important performance advantages to the system. These advantages come from the appearance of the processed data blocks in system memory without consuming any system resources or bus time.

The advantages of this approach are best demonstrated by comparing it to alternative means of adding a real-time task to an existing real-time system. In this case, the new task requires additional CPU, memory, and I/O resources.

The traditional multiprocessor approach of Fig. 2a expands CPU resources in one of two ways: software utilization of reserve capacity in the existing processor, or adding another processor. In either case, memory and I/O increments generally will be required.

The primary disadvantages of this approach are the increased complexity of the system software and the increased load on the system bus. Both will slow the existing real-time system unless it has been designed with adequate reserve. The system bus must also provide sufficient capacity for the incremental memory-execution and data-transfer operations. This additional bus load will also require that the primary real-time task can tolerate CPU delays due to bus contention.

The intelligent-controller approach of Fig. 2b has gained widespread use since the advent of the micropro-



4. Memory mapping. The variable system memory addresses are always mapped into the on-board address of 8000HEX, providing software independence for the subsystem and the host.

cessor. This approach combines the CPU and I/O increments onto a single module that usually includes direct-memory-access transfer logic. In some cases the execution memory for the CPU is included.

This approach lessens the bus-loading problem since the I/O data transfers and some memory-execution cycles take place off the system bus. However, both CPUs' programs will have to tolerate delays caused by increased bus contention. Increased software sophistication is the primary disadvantage of this approach, much as with the multiprocessing approach of Fig. 2a.

The intelligent-slave approach of Fig. 2c can be viewed as a logical extension to the intelligent-controller approach. Combining the CPU, I/O, and memory increments creates a single module that has a minimal impact on the existing system software and bus loading. What's more, the subsystem can operate at full capacity without regard to other system activity. It can be programmed outside the primary system and then added with minimal impact on the system software or performance.

A limitation of the approach is the inability of the subsystem to transfer data into portions of the system memory space that reside off its board. This problem is minimized by the ability of the controller's RAM to serve as a substantial portion of the entire memory space addressable by the system. In this light, the on-board processor can be viewed as having a DMA capability limited to a portion of the system's address space.

In a system with more than one of the new controllers, the system CPU handles any data that must be trans-

TABLE: EIA RS. 232-C SIGNALS PROVIDED AND SUPPORTED

Carrier detect	Receive clock
Clear to send	Receive data
Data set ready	Ring indicator
Data terminal ready	Transmit clock
Request to send	Transmit data

ferred from one to another. Applications involving the transfer of large blocks of data would be best served by a central block-transfer device elsewhere on the bus.

The advantages offered by the new approach in this example of adding onto an existing system are just as applicable to a ground-up design. This modular approach to configuring real-time multiprocessing systems simplifies hardware and software design, as well as system integration.

While the primary design objective of the new architecture is operation in a multiprocessing system, it can provide significant utility as stand-alone processors. Thus these controllers incorporate a second mode of operation called the limited bus-master mode.

In this mode of operation the new controller can be used like a single-board computer as long as it is the system's only master of the bus. It can be connected to standard memory or I/O expansion boards to enhance its capability. It can even be used to drive other such controllers as long as they are used in the subsystem mode. This dual operational mode will allow the new controllers to serve a broad range of applications.

Communications first

Communications applications present complex processing requirements and an inherent real-time nature, so it is logical that a communications processor be the first of these new controllers to be marketed. The iSBC 544 intelligent communications controller can serve as a flexible front end to an iSBC system or as a cost-effective stand-alone processor configured as a terminal cluster or line concentrator. Its design (Fig. 3) incorporates an 8085A CPU, 16 kilobytes of dual-ported dynamic RAM, 4 kilobytes of PROM, programmable interrupt control, three interval timers, four programmable baud-rate generators, four synchronous/asynchronous buffered serial I/O channels, and a 10-bit parallel interface compatible with a Bell 801 automatic calling unit.

The dual-port memory block basically consists of the 16-kilobyte bank of random-access memory, which is accessible from either the system bus or the on-board processor through the dual-port controller. This memory block provides the primary means of communication between the system and the on-board 8085A. The port to the memory, which looks to the system bus like any other RAM card belonging to the system, features full 20-bit

addressing and a typical access time of 600 nanoseconds.

The interface's address-decode logic allows switching of the base address of the iSBC 544 to any 4-kilobyte boundary in the host system's address space. In addition, the user may reserve 8, 12, or 16 kilobits of the 544's memory for use by the on-board processor only. This reserved memory is not accessible from the system bus and does not occupy any system address space. The only restriction is that all of the unreserved memory reside in the same 64-K address page of the system memory.

This memory division can be a significant advantage in large 8-bit microcomputer systems. Only that portion of the controllers' memory needed to pass data between CPUs must be made accessible to the system. The remaining buffer and execution memory does not consume any system address space.

The net result is an increase in the system's overall memory capacity. For example, a microcomputer system that would usually be limited to 64 kilobytes of memory has a total memory capacity of over 190 kilobytes when driving seven 544s.

Address maps and interrupts

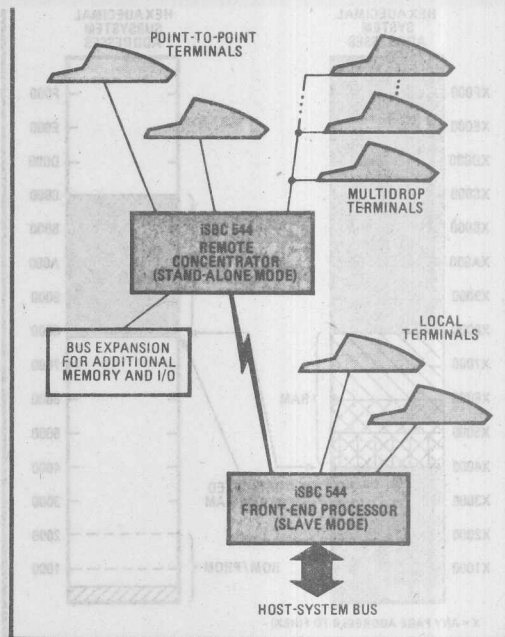
To the on-board processor, the base address of its memory is fixed at 8000HEX. Furthermore, all on-board addresses are fixed, so that multiple 544s operating on the same system bus can be running identical programs regardless of their base address on that bus. This capability necessitated the address-mapping logic to transform addresses from the system bus into the equivalent in the on-board address space starting at location 8000HEX (Fig. 4).

The address-mapping logic also implements the flag-interrupt feature. It provides an interrupt to the on-board processor whenever a byte is written into the 544's base address from the system bus, and a read from the on-board processor to the base address clears the interrupt. Since each 544 in a system has a different base address in that system's RAM, it also has a unique interrupt. This flag-interrupt capability is a key element in establishing a protocol for communications between the host CPU and the subsystems' processors.

The dual-port control logic is responsible for resolving contention over access to the memory and is designed to optimize the performance of the subsystem CPU. Unless the system bus has initiated a memory cycle before the on-board processor requests memory, that CPU runs at full speed. The maximum delay that can be encountered is one memory cycle. The arbitration logic actually reserves the memory for the on-board processor before it generates the necessary commands. This advance reserving guarantees that the on-board CPU will suffer minimum intervention from system bus accesses.

When the iSBC 544 is used in the stand-alone limited bus-master mode, the dual-port logic is disabled and the bus interface buffers are turned around to drive onto the bus. This reversal allows the on-board central-processing unit access to the memory of other subsystems or I/O expansion boards on the system bus.

The dedicated computer is built with an 8085A CPU operating at 2.76 megahertz, between 2 and 4 kilobytes of PROM and ROMs or 8 kilobytes of ROM using 2332



5. Communications applications. Two typical applications of the new iSBC 544 would be as a front-end communications processor to a microcomputer system and as a remote concentrator to a series of point-to-point or multidrop connected terminals.

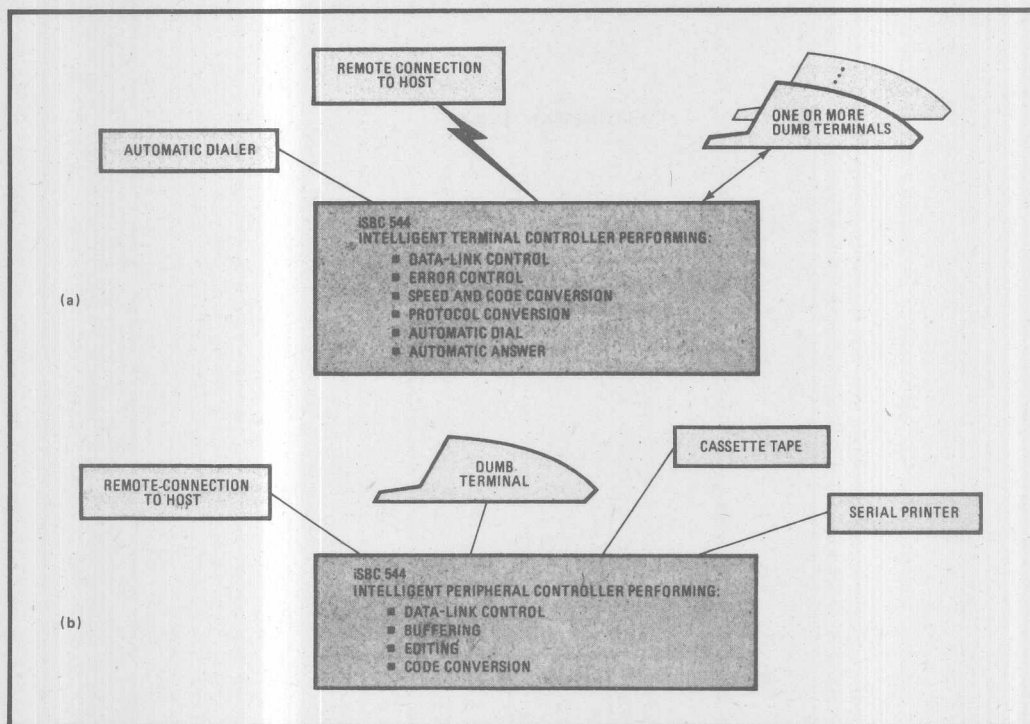
mask-programmable parts, 256 bytes of static RAM, two 16-bit and one 14-bit interval timers, and a 8259 programmable interrupt controller for individual receive or transmit interrupt inputs for each serial port.

Special command-decode logic was added to the CPU to allow it to operate at maximum speed independent of other system activity. There are 21 sources of interrupt on the 544, including the separate transmit and receive interrupts for each port and separate timer interrupts. In addition to receiving an interrupt from the system, the 544 can also send an interrupt to the system bus via the 8085A's serial-output data line.

Since this controller is intended for communications applications, latched interrupts are provided directly to the CPU for loss of carrier and ring indicator for all four I/O ports. The ring-indicator and carrier-detect lines can also be monitored through the parallel port.

Dedicated I/O

The dedicated-I/O section of the 544 provides a high degree of flexibility and programmability. This results primarily from the inclusion of four 8251A universal synchronous/asynchronous receiver/transmitters. These devices are programmable for synchronous or asynchronous mode, character size, parity bits, stop bits, and baud rates. Data, clocks, and control lines are buffered with RS-232-C-compatible drivers and receivers to four 26-pin card-edge connectors. Each port is configured as a data-terminal interface, but may be converted to a



6. From slave to master. In its stand-alone mode, the 544 can operate as a bus master and be configured as an intelligent terminal controller connecting dumb terminals to a data link (a) or as a peripheral controller connecting RS-232-C-compatible units to the terminal (b).

data-set interface by changing a single jumper-plug assembly. The ports support most RS-232-C signals (those that are listed in the table).

A programmable baud-rate generator is also provided for each port. The range of baud rates available is 75 to 56 kilobits per second. The generators are implemented with 8253 programmable interval timers, which receive a jumper-selectable input frequency of 1.84 or 1.23 MHz. In addition, one of the CPU's interval timers can be converted to baud-rate operation and jumpered to any port to provide it with split-speed operation.

The 544 also provides a parallel port with four RS-232-C buffered input lines and six RS-232-C buffered output lines. This port is configured to interface to most automatic calling units but may be used as a general-purpose I/O port. It is implemented with an 8155 programmable peripheral interface that also provides the 256 bytes of static RAM and the 14-bit timer.

Applications

A likely common use of the 544 as a subsystem is as a front-end processor or terminal multiplexer (Fig. 5) in an iSBC system. The 544 performs all communications-related functions such as format control, code conversion, data-link control, error checking, data compression, and protocol management. It can handle multiple protocols, line speeds, and data formats.

All the system processor sees are the processed data

blocks that appear in system memory. An automatic dialer could be added to provide a dial-up connection to a host processor or network.

Also shown in Fig. 5 is another 544 used in its limited bus-master mode as a remote concentrator and terminal controller. The line and memory capacity of the remote concentrator can be increased by the addition of standard iSBC memory and I/O expansion boards.

The intelligent-terminal controller shown in Fig. 6a is a prime example of a 544 used in the stand-alone mode. It can connect one or more dumb terminals to a data link and provide the necessary buffering, code conversion, and data-link control. It could also connect a terminal that happens to communicate in a different protocol to a new network or to more than one network.

The iSBC 544's multiple serial lines do not have to be used for communications. They can also be used to connect RS-232-C-compatible peripherals to the terminal (Fig. 6b). In this configuration, the 544 can provide message editing and formatting, bulk storage, and hard-copy output.

As this last application suggests, the 544 is the vanguard of a family of intelligent I/O controllers that will add tremendous increases in throughput and versatility to the iSBC line of single-board computers. The basic architecture will simplify the task of developing multiprocessing hardware and software solutions that will overcome throughput limitations. □

and Signal Conditioning Boards

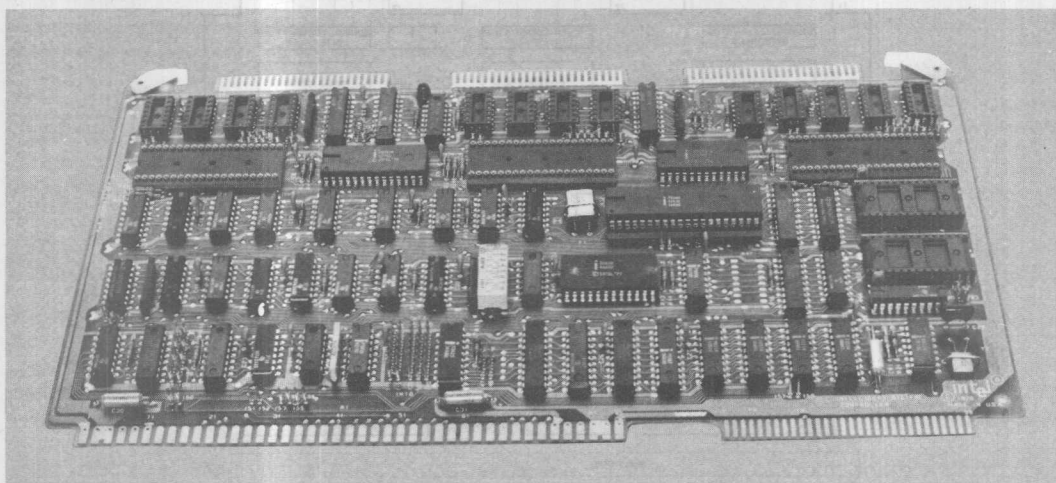
9



iSBC 569 INTELLIGENT DIGITAL CONTROLLER

- Single board digital I/O controller with up to four microprocessors to share the digital input/output signal processing
- 3 MHz 8085A central control processor
- Three sockets for 8041/8741A Universal Peripheral Interface (UPI-41A) for distributed digital I/O processing, such as:
 - Industrial signal processor (iSBC 941)
 - Custom programmed 8041A/8741A
- Three operational modes
 - Stand-alone digital controller
 - MULTIBUS master
 - Intelligent slave (slave to MULTIBUS master)
- 2K bytes of dual port static read/write memory
- Sockets for up to 8K bytes of Intel 2758, 2716, 2732 erasable programmable read only memory
- 48 programmable parallel I/O lines with sockets for interchangeable line drivers or terminators
- Three programmable counters
- 12 levels of programmable interrupt control
- Single + 5V supply
- MULTIBUS standard control logic compatible with optional iSBC 80 and iSBC 86 CPU, memory, and I/O expansion boards

The Intel iSBC 569 Intelligent Digital Controller is a single board computer (8085A based) with sockets for three 8041A/8741A Universal Peripheral Interface chips (UPI-41A). The I/O processing algorithm may be tailored to application requirements using designer selected combinations of standard Intel industrial signal processors (e.g., iSBC 941) or user programmed UPI-41A processors. These devices may be used to offload the 8085A processor from time consuming tasks such as pulse counting, event sensing, and parallel or serial digital I/O data formatting with error checking and handshaking. The iSBC 569 board is a complete digital controller with up to four processors on a single 6.75 inches x 12.00 inches (17.15cm x 30.48cm) printed circuit board. The 8085A CPU, system clock, read/write memory, non-volatile memory, priority interrupt logic, programmable timers, MULTIBUS control and interface logic, optional UPI processors and optional line driver and terminators all reside on one board.



FUNCTIONAL DESCRIPTION

Intelligent Digital Controller

Three modes of operation — the iSBC 569 Intelligent Digital Controller is capable of operating in one of three modes; stand alone controller, bus master, or intelligent slave.

Stand alone controller — the iSBC 569 board may function as a stand alone, single board controller with CPU, memory, and I/O elements on a single board. Five volt (+5VDC) only operation allows configuration of low cost controllers with only a single power supply voltage. The on-board 2K bytes RAM and up to 16K bytes ROM/EPROM, as well as the assistance of three UPI-41A processors, allow significant digital I/O control from a single board.

Bus master — in this mode of operation, the iSBC 569 controller may interface with and control iSBC expansion memory and I/O boards, or even other iSBC 569 Intelligent Digital Controllers configured as intelligent slaves (but no additional bus masters).

Intelligent slave — the iSBC 569 controller can perform as an intelligent slave to any 8- or 16-bit MULTIBUS master CPU by offloading the master of digital control related tasks. Preprocessing of data for the master is controlled by the on-board 8085A CPU which coordinates up to three UPI-41A processors. Using the iSBC 569 board as an intelligent slave, multi-channel digital control can be managed entirely on-board, freeing a system master to perform other system functions. The dual port RAM memory allows the iSBC 569 controller to process and store data without MULTIBUS memory contention.

Simplified Programming

By using Intel UPI-41A processors for common tasks such as counting, sensing change of state, printer control and keyboard scanning/debouncing, the user frees up time to

work on the more important application programming of machine or process optimization. Controlling the Intel UPI-41A processors becomes a simple task of reading or writing command and data bytes to or from the data bus buffer register on the UPI device. Programming the iSBC 941 Industrial Digital Processor to produce a pulse output, for example, is as simple as sending command and parameter bytes indicating initialization, pulse output selection, period and delay parameters, followed by a command to begin execution.

Central Processing Unit

A powerful Intel 8085A 8-bit CPU, fabricated on a single LSI chip, is the central processor for the iSBC 569™ controller. The six general purpose 8-bit registers may be addressed individually or in pairs, providing both single and double precision operations. The program counter can address up to 64K bytes of memory using iSBC expansion boards. The 16-bit stack pointer controls the addressing of an external stack. This stack provides sub-routine nesting bounded only by memory size. The minimum instruction execution time is 1.30 microseconds. The 8085A CPU is software compatible with the Intel 8080A CPU.

Bus Structure

The iSBC 569 Intelligent Digital Controller utilizes a triple bus architecture concept. An internal bus is used for on-board memory and I/O operations. A MULTIBUS interface is available to provide access for all external memory and I/O operations. A dual port bus with controller enables access via the third bus to 2K bytes of static RAM from either the on-board CPU or a system master. Hence, common data may be stored in on-board memory and may be accessed either by the on-board CPU or by system masters. A block diagram of the iSBC 569 functional components is shown in Figure 1.

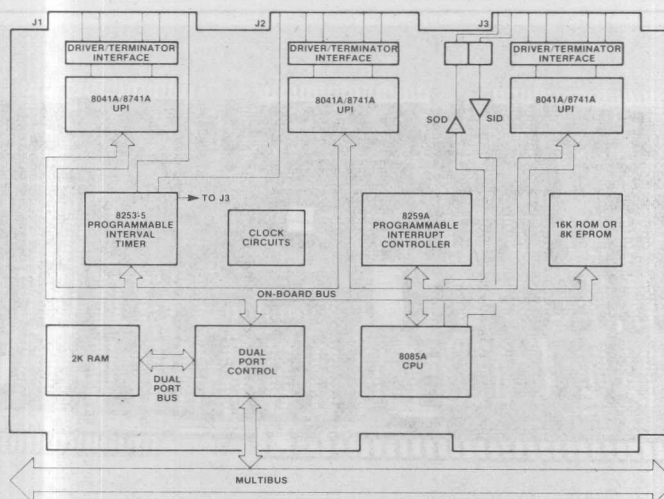


Figure 1. iSBC 569 Intelligent Digital Controller Block Diagram

RAM Capacity

The iSBC 569 board contains 2K bytes of read/write memory using Intel 2114 static RAMs. RAM accesses may occur from either the iSBC 569 controller or from any other bus master interfaced via the MULTIBUS system bus. The iSBC 569 board provides addressing jumpers to allow the on-board RAM to reside within a one megabyte address space when accessed via the system bus. In addition, a switch is provided which allows the user to reserve a 1K byte segment of on-board RAM for use by the 8085A CPU. This reserved RAM space is not accessible via the system bus and does not occupy any system address space.

EPROM/ROM Capacity

Two sockets for up to 16K bytes of nonvolatile read only memory are provided on the iSBC 569 board. Nonvolatile memory may be added in 1K-byte increments up to a maximum of 2K bytes using Intel 2758 erasable and electrically reprogrammable ROMs (EPROMs); in 2K-byte increments up to a maximum of 4K bytes using Intel 2316 ROMs or 2716 EPROMs; in 4K byte increments up to 8K bytes maximum using Intel 2732 EPROMs; or in 8K-byte increments up to 16K bytes maximum using Intel 2364 ROMs (both sockets must contain same type ROM/EPROM). All on-board ROM/EPROM operations are performed at maximum processor speed.

Universal Peripheral Interfaces (UPI-41A)

The iSBC 569 Intelligent Digital Controller board provides three sockets for user supplied Intel 8041A/8741A Universal Peripheral Interface (UPI-41A) chips. Sockets are also provided for the associated line drivers and terminators for the UPI I/O ports. The UPI-41A processor is a single chip microcomputer containing a CPU, 1K bytes of ROM (8041A) or EPROM (8741A), 64 bytes of RAM, 16 programmable I/O lines, and an 8-bit timer/event counter. Special interface registers included in the chip allow the UPI-41A processor to function as a slave processor to the iSBC 569 controller board's 8085A CPU. The UPI processor allows the user to specify algorithms for controlling peripherals directly thereby freeing the 8085A for other system functions. For additional information, including UPI-41A instructions, refer to the UPI-41 User's Manual (Manual No. 9800504).

Industrial Digital Processor (ISBC 941)

The iSBC 941 Industrial Digital Processor is a 40-pin DIP device which provides the user with easy-to-use processing of digital input and output signals desired in many industrial automation environments. One of nine digital I/O functions can be selected at any one time for measuring, counting, or controlling up to 16 separate I/O lines. An additional eight utility commands allow reading or setting the condition of unused I/O lines. Simplex serial input and output modes can assemble or disassemble bytes transmitted asynchronously over TTL lines, including insertion and deletion of start/stop bits. The iSBC 941 processor plugs into any of the three UPI-41A sockets on the iSBC 569 board. Simple programming commands from the master 8085A processor can thus implement up to 48 lines of preprocessed digital I/O signals. For specific commands and further information, refer to the iSBC 941 Data Sheet in this document.

Programmable Timers

The iSBC 569 Intelligent Digital Controller board provides three independently programmable interval timer/counters utilizing one Intel 8253 Programmable Interval Timer (PIT). The Intel 8253 PIT provides three 16-bit BCD or binary interval timer/counters. Each timer may be used to provide a time reference for each UPI™ processor or for a group of UPI processors. The output of each timer also connects to the 8259A Programmable Interrupt Controller (PIC) providing the capability of timed interrupts. All gate inputs, clock inputs, and timer outputs of the 8253 PIT are available at the I/O ports for external access.

Timer Functions — In utilizing the iSBC 569 controller, the systems designer simply configures, via software, each timer to meet systems requirements. The 8253 PIT modes are listed in Table 1. The contents of each counter may be read at any time during system operation with simple read operations for event counting applications. The contents of each counter can be read "on-the-fly" for time stamping events or time clock referenced program initiations.

Table 1. 8253 Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated.
Programmable one-shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low-going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge on counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counting "window" has been enabled or an interrupt may be generated after N counts occur in the system.

Interrupt Capability

The ISBC 569 Intelligent Digital Controller provides interrupt service for up to 12 interrupt sources. Any of the 12 sources may interrupt the on-board processor. Four interrupt levels are handled directly by the 8085A CPU and eight levels are serviced from an Intel 8259A Programmable Interrupt Controller (PIC) routing an interrupt request output to the INTR input of the 8085A.

8085A Interrupt — Each of four direct 8085A interrupt inputs has a unique vector memory address. An 8085A jump instruction at each of these addresses then provides software linkage to interrupt service routines located independently anywhere in the memory.

8259A Interrupts — The eight interrupt sources originate from both on-board controller functions and the system bus:

UPI-41A Processors — one interrupt from each of three UPI processor sockets.

8253 PIT — one interrupt from each of three timer outputs.

MULTIBUS System Bus — one of eight MULTIBUS interrupt lines may be jumpered to either of two 8259A PIC interrupt inputs.

Programmable Reset — The ISBC 569 Intelligent Digital Controller board has a programmable output latch used to control on-board functions. Three of the outputs are connected to separate UPI-41A RESET inputs. Thus, the user can reset any or all of the UPI-41A processors under software control. A fourth latch output may be used to generate an interrupt request onto the MULTIBUS interrupt lines. A fifth latch output is connected to a light-emitting diode which may be used for diagnostic purposes.

Expansion Capabilities

When the ISBC 569 controller is used as a single board digital controller, memory and I/O capacity may be expanded using Intel MULTIBUS compatible expansion boards. In this mode, no other bus masters may be in the system. Memory may be expanded to a 64K byte capacity by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding I/O expansion boards. Multiple ISBC 569 boards may be included in an

expanded system using one ISBC 569 Intelligent Digital Controller as the system master and additional controllers as intelligent slaves.

Intelligent Slave Programming

When used as an intelligent slave, the ISBC 569 controller appears as an additional RAM memory module. System bus masters communicate with the ISBC 569 board as if it were just an extension of system memory. To simplify this communication, the user has been given some specific tools:

Flag Interrupt — The Flag Interrupt is generated any time a write command is performed by an off-board CPU to the first location of ISBC 569 RAM. This interrupt provides a means for the master CPU to notify the ISBC 569 controller that it wished to establish a communications sequence. The flag interrupt is cleared when the on-board processor reads the first location of its RAM. In systems with more than one intelligent slave, the flag interrupt provides a unique interrupt to each slave outside the normal MULTIBUS interrupt lines (INT0-INT7).

RAM — The on-board 2K byte RAM area that is accessible to both an off-board CPU and the on-board 8085A may be configured for system access on any 2K boundary.

MULTIBUS Interrupts — The third tool to improve system operation as an intelligent slave is access to the MULTIBUS interrupt lines. The ISBC 569 controller can both respond to interrupt signals from an off-board CPU, and generate an interrupt to the off-board CPU via the system bus.

System Development Capability

Software development for the ISBC 569 Intelligent Digital Controller board is supported by the Intellec® Microcomputer Development System including a resident macroassembler, text editor, system monitor, a linker, object code locator, and Library Manager. In addition, both PL/M and FORTRAN language programs can be compiled to run on the ISBC 569 board. A unique in-circuit emulator (ICE-85™) option provides the capability of developing and debugging software directly on the ISBC 569 board. This greatly simplifies the design, development, and debug of ISBC 569 system software.

SPECIFICATIONS

8085A CPU

Word Size — 8, 16 or 24 bits

Cycle Time — 1.30 μ sec \pm .1% for fastest executable instruction; i.e., four clock cycles.

Clock Rate — 3.07 MHz \pm .1%

System Access Time

Dual port memory — 725 nsec

Memory Capacity

On-board ROM/EPROM — 2K, 4K, 8K, or 16K bytes of user installed ROM or EPROM

On-board RAM — 2K bytes of static RAM. Fully

accessible from on-board 8085A. Separately addressable from system bus.

Off-board expansion — up to 64K bytes of EPROM/ROM or RAM capacity.

I/O Capacity

Parallel-Timers — Three timers, with independent gate input, clock input, and timer output user-accessible. Clock inputs can be strapped to an external source or to an on-board 1.3824 MHz reference. Each timer is connected to a 8259A Programmable Interrupt Controller and may also be optionally connected to UPI processors.

UPI-I/O — Three UPI-41A interfaces, each with two 8-bit I/O ports plus the two UPI Test Inputs. The 8-bit ports are user-configurable (as inputs or outputs) in groups of four.

Serial — 1 TTL compatible serial channel utilizing SID and SOD lines of on-board 8085A CPU

On-Board Addressing

All communications to the UPI-41A processors, to the programmable reset latch, to the timers, and to the interrupt controller are via read and write commands from the on-board 8085A CPU.

Memory Addressing

On-board ROM/EPROM — 0-07FF (using 2758 EPROMs); 0-OFFF (using 2716 EPROMs or 2316 ROMs); 0-1FFF (using 2732 EPROMs); 0-3FFF (using the 2364 ROMs)

On-board RAM — 8000-87FF System access — any 2K increment 00000-FF800 (switch selectable); 1K bytes may be disabled from bus access by switch selection.

I/O Addressing

Source	Addresses
8253	0E0H-0E3H
UPI0	0E4H-0E5H
UPI1	0E6H-0E7H
UPI2	0E8H-0E9H
PROGRAMMABLE RESET	0EAH-0EBH
8259A	0ECH-0EDH

Timer Specifications

Input frequencies — jumper selectable reference

Internal: 1.3824 MHz \pm .1% (.723 μ sec, nominal)

External: User supplied (2 MHz maximum)

Output Frequencies (at 1.3824 MHz)

Function	Min'	Max'
Real-time interrupt interval	1.45 μ sec	47.4 msec
Rate Generator (frequency)	21.09 Hz	691.2 KHz
1. Single 16-bit binary count		

Interfaces

MULTIBUS™ Interface — All signals compatible with iSBC and MULTIBUS architecture

Parallel I/O — All signals TTL compatible

Interrupt Requests — All TTL compatible

Timer — All signals TTL compatible

Serial I/O — All signals TTL compatible

Connectors

Interface	Pins (qty)	Centers (in.)	Mating Connectors
Bus	86	0.156	Viking 3KH43/9AMK12
Parallel I/O	50	0.1	3M 3415-000 or TI H312125

Physical Characteristics

Width — 30.48 cm (12.00 inches)

Depth — 17.15 cm (6.75 inches)

Thickness — 1.27 cm (0.50 inch)

Weight — 3.97 gm (14 ounces)

Electrical Characteristics

DC Power Requirements — +5V @ 2.58A with no optional devices installed. For each 8741A add 135 mA. For each 220/330 resistor network, add 60 mA. Add the following for each EPROM/ROM installed.

Type	+5.0V Current Requirement	
	1ROM	2ROMS
2758	100 mA	125 mA
2716	100 mA	125 mA
2316E	120 mA	240 mA
2732	40 mA	55 mA
2364	40 mA	55 mA

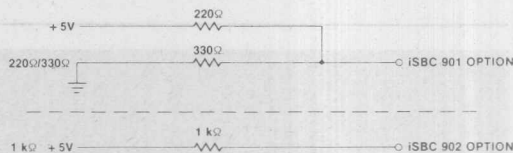
Line Drivers and Terminators

I/O Drivers — The following line drivers are all compatible with the I/O driver sockets on the iSBC 569 Intelligent Digital Controller.

Driver	Characteristic	Sink Current (mA)
7438	I,OC	48
7437	I	48
7432	NI	16
7426	I,OC	16
7409	NI,OC	16
7408	NI	16
7403	I,OC	16
7400	I	16

Note — I = inverting; NI = non-inverting; OC = open collector

I/O Terminators — 220 Ω /330 Ω divider or 1 k Ω pullup



Environmental Characteristics

Operating Temperature — 0°C to 55°C (32°F to 131°F)

Relative Humidity — To 90% without condensation

Reference Manuals

502180 — iSBC 569 Intelligent Digital Controller Board Hardware Reference Manual (NOT SUPPLIED)

503100 — iSBC 941 Digital Signal Processor User's Guide (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 569	Intelligent Digital Controller



iSBC 556 OPTICALLY ISOLATED I/O BOARD

- iSBC 80 and MULTIBUS compatible
- Up to 48 digital optically isolated input/output data lines
- Choice of
 - 24 fixed input lines
 - 16 fixed output lines
 - 8 programmable lines
- Provisions for plug-in, optically isolated receivers, drivers, and terminators
- Voltage/current levels
 - Input up to 48V
 - Output up to 30V, 60 mA
- Common interrupt for up to 8 sources
- +5V supply only

The iSBC 556 Optically Isolated I/O Board provides 48 digital input/output lines with isolation between process application or peripheral device and the iSBC 80 series single board computers. The iSBC 556 contains two 8255A programmable interface devices, and sockets for user supplied optically isolated drivers, receivers, and input resistor terminators, together with common interrupt logic and iSBC 80 bus interface logic. Input signals can be single-ended or differential types with user defined input range (resistor terminator and opto-isolated receiver selection), allowing flexibility in design of voltage and threshold levels. The output allows user selection of Opto-Isolated Darlington Pair which can be used as an output driver either as an open collector or current switch.

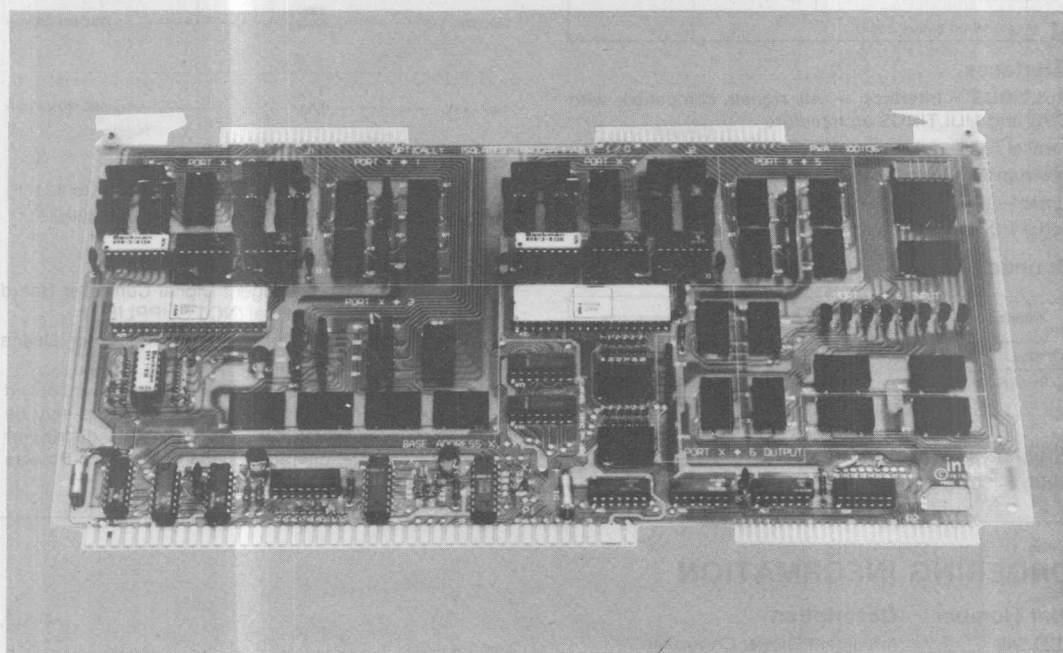


Table 1. I/O Ports Opto-Isolator Receivers, Drivers, and Terminators

Port No. X = I/O Base Address	Type of I/O	Lines (qty)	Resistor Terminator Pac-Rp 16-Pin DIP Bourns 4116R-00 or Equivalent	Dual Opto-Isolator 8-Pin DIP Monsanto MC T66 or Equivalent	Driver 7438 or Equivalent	Pull-Up iSBC® 902
X+0	Input	8	1	4	—	—
X+1	Output	8	—	—	—	—
X+2	Input/Control	8	1	—	—	—
X+4	Input	8	1	4	—	—
X+5	Output	8	—	—	—	—
X+6	Input/Control	8	1 if input	—	2 if output	2 if input
X+7	Output	8	—	—	—	—

SPECIFICATIONS

Number of Lines

24 input lines

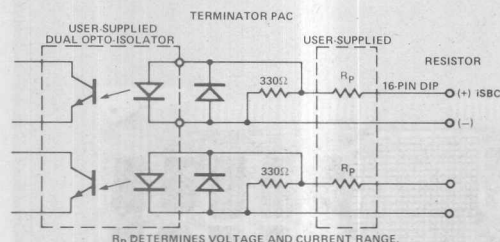
16 output lines

8 programmable lines: 4 input — 4 output

I/O Interface Characteristics

Line-to-Line Isolation — 235V DC or peak AC

Input/Output Isolation — 500V DC or peak AC



Bus Interface Characteristics

All data address and control commands are iSBC 80 bus compatible.

I/O Addressing

Port	8255 #1			Control	8255 #2			Control
	A	B	C		A	B	C	
Address	X+0	X+1	X+2	X+3	X+4	X+5	X+6	X+7

Where:

base address is from 00H to 1FH (jumper selectable)

ORDERING INFORMATION

Part Number Description

SBC 556 Optically Isolated I/O Board

Connectors

Interface	Pins (qty)	Centers		Mating Connectors
		in.	cm	
P1 iSBC bus	86	0.156		Viking 3KH43/9AMK12
J1 16 fixed input & 8 fixed output lines	50	0.1		3M 3415-000 or TI M312125
J2 8 fixed output, 8 fixed output, & 8 programmable input/output lines	50	0.1		3M 3415-000 or TI M312125

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.50 in. (1.27 cm)

Weight — 12 oz (397.3 gm)

Electrical Characteristics

Average DC Current

V_{CC} = +5V ± 5%, 1.0A without user supplied isolated receiver/driver

I_{CC} = 1.6A max with user supplied isolator receiver/driver

Environmental Characteristics

Temperature — 0°C to 55°C

Relative Humidity — 0 to 90%, non-condensing

Reference Manual

502170 — iSBC 556 Hardware Reference Manual (NOT SUPPLIED)

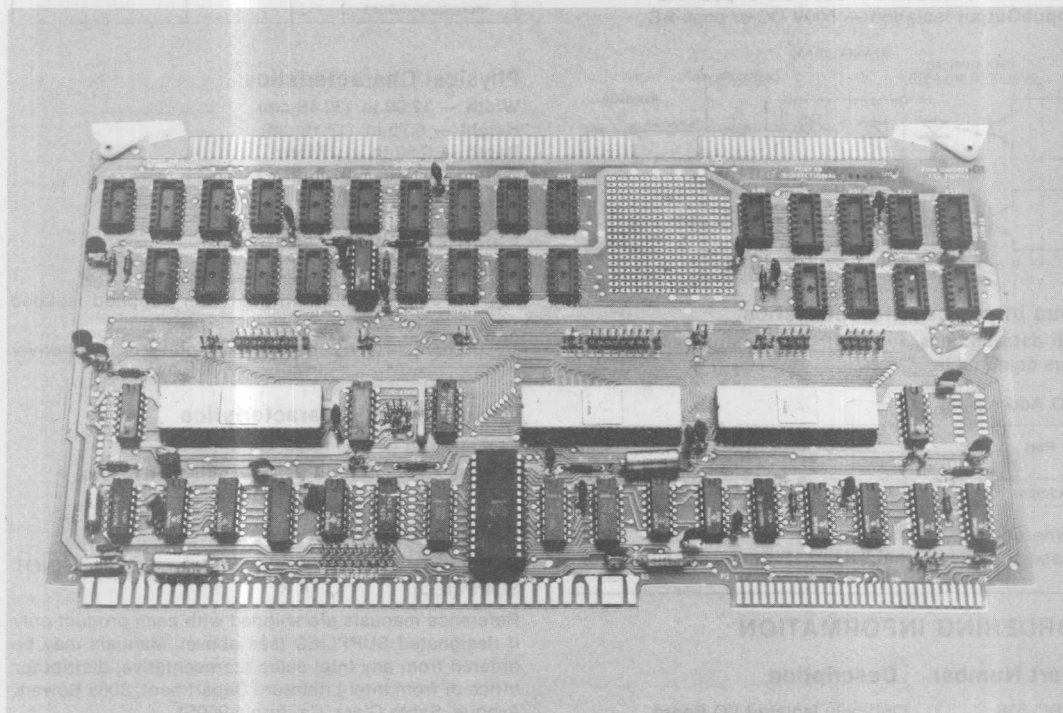
Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.



iSBC 519 (or pSBC 519*) PROGRAMMABLE I/O EXPANSION BOARD

- iSBC I/O expansion via direct MULTIBUS Interface
- 72 programmable I/O lines with sockets for interchangeable line drivers and terminators
- Jumper selectable I/O port addresses
- Jumper selectable 0.5, 1.0, 2.0, or 4.0 ms interval timer
- Eight maskable interrupt request lines with priority encoded and programmable interrupt algorithms

The iSBC 519 Programmable I/O Expansion Board is a member of Intel's complete line of iSBC memory and I/O expansion boards. The iSBC 519 interfaces directly to any iSBC single board computer via the system bus to expand input and output port capacity. The iSBC 519 provides 72 programmable I/O lines. The system software is used to configure the I/O lines to meet a wide variety of peripheral requirements. The flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. Address selection is accomplished by using wire-wrap jumpers to select one of 16 unique base addresses for the input and output ports. The board operates with a single +5V power supply.



* Same product, manufactured by Intel Puerto Rico, Inc.

FUNCTIONAL DESCRIPTION

The 72 programmable I/O lines on the iSBC 519 are implemented utilizing three Intel 8255 programmable peripheral interfaces. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports indicated in Table 1. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. The 72 programmable I/O lines and signal ground lines are brought out to three 50-pin edge connectors that mate with flat, round, or woven cable.

Interval Timer

Typical I/O read access time is 350 nanoseconds.

Typical I/O read/write cycle time is 450 nanoseconds. The interval timer provided on the iSBC 519 may be used to generate real time clocking in systems requiring the periodic monitoring of I/O functions. The time interval is derived from the constant clock (BUS CCLK) and the timing interval is jumper selectable. Intervals of 0.5, 1.0, 2.0, and 4.0 milliseconds may be selected when an iSBC single board computer is used to generate the clock. Other timing intervals may be generated if the user provides a separate constant clock reference in the system.

Eight-Level Vectored Interrupt

An Intel 8259 programmable interrupt controller (PIC) provides vectoring for eight interrupt levels. As shown in Table 2, a selection of three priority processing algorithms is available to the system designer so that the

Table 1. Input/Output Port Modes of Operation

Ports	Lines (qty)	Mode of Operation				Bidirectional	Control
		Unidirectional					
		Input		Output			
		Unlatched	Latched & Strobed	Latched	Latched & Strobed		
1,4,7	8	X	X	X	X	X	
2,5,8	8	X	X	X	X		
3,6,9	4	X		X			X ^{1,2,3}
	4	X		X			X ^{1,2,3}

Notes

1. Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.
2. Part of port 6 must be used as a control port when either port 4 or port 5 are used as a latched and strobed input or a latched and strobed output port or port 4 is used as a bidirectional port.
3. Part of port 9 must be used as a control port when either port 7 or port 8 are used as a latched and strobed input or a latched and strobed output port or port 7 is used as a bidirectional port.

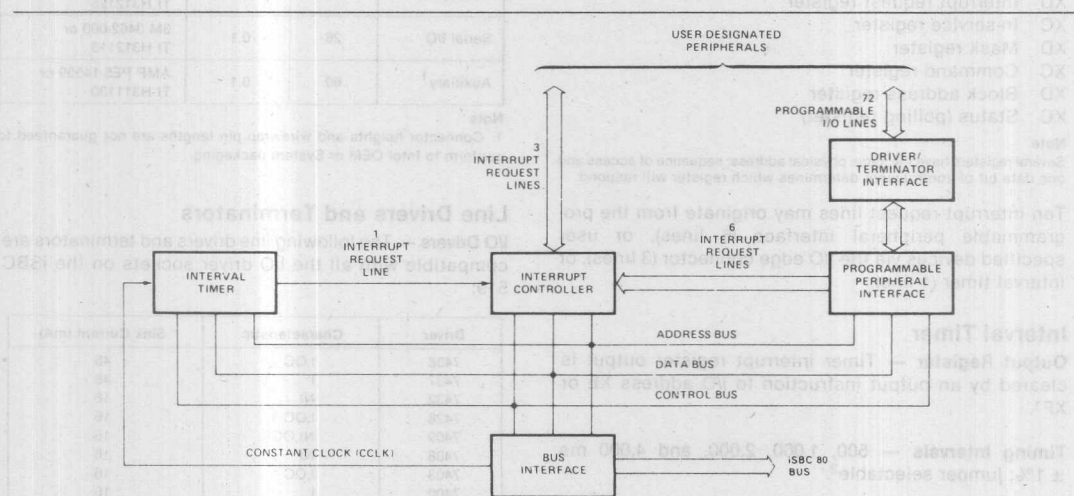


Figure 1. iSBC 519 Programmable I/O Expansion Board Block Diagram

Algorithm	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels are based in sequence numerically on this assignment.

manner in which requests are serviced may be configured to match system requirements. Priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from the programmable parallel I/O interfaces, the interval timer, or direct from peripheral equipment. The PIC then determines which of the

whether this request is of higher priority than the level currently being serviced, and if appropriate, issues an interrupt to the system master. Any combination of interrupt levels may be masked through storage, via software, of a single byte to the interrupt mask register of the PIC.

Interrupt Request Generation — Interrupt requests may originate from 10 sources. Six jumper selectable interrupt requests can be automatically generated by the programmable peripheral interfaces when a byte of information is ready to be transferred to the system master (i.e., input buffer is full) or a character has been transmitted (i.e., output data buffer is empty). Three interrupt request lines may be interfaced to the PIC directly from user designated peripheral devices via the I/O edge connectors. One interrupt request may be generated by the interval timer.

Bus Line Drivers — The PIC interrupt request output line may be jumper selected to drive any of the nine interrupt lines on the MULTIBUS. Any of the on-board request lines may also drive any interface interrupt line directly via jumpers and buffers on the board.

SPECIFICATIONS

Addressing

Port	1	2	3	8255 No. 1 Control	4	5	6	8255 No. 2 Control	7	8	9	8255 No. 3 Control
Address	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB

Interrupts

Register Addresses (hex notation, I/O address space)

XD	Interrupt request register
XC	In-service register
XD	Mask register
XC	Command register
XD	Block address register
XC	Status (polling) register

Note

Several registers have the same physical address; sequence of access and one data bit of control word determines which register will respond.

Ten interrupt request lines may originate from the programmable peripheral interface (6 lines), or user specified devices via the I/O edge connector (3 lines), or interval timer (1 line).

Interval Timer

Output Register — Timer interrupt register output is cleared by an output instruction to I/O address XE or XF1.

Timing Intervals — 500, 1,000, 2,000, and 4,000 ms $\pm 1\%$; jumper selectable².

Notes

1. X is any hex digit assigned by jumper selection.

2. Assumes constant clock (CCLK) frequency of 9.216 MHz $\pm 1\%$.

Interfaces

Bus — All signals TTL compatible

Parallel I/O — All signals TTL compatible

Interrupt Requests — All TTL compatible

Connectors

Interface	Pins (qty)	Centers (in.)	Mating Connectors
Bus	86	0.156	Viking 3KH43/9AMK12
Parallel I/O	50	0.1	3M 3415-000 or TI H312125
Serial I/O	26	0.1	3M 3462-000 or TI H312113
Auxiliary ¹	60	0.1	AMP PE5-14559 or TI H311130

Note

1. Connector heights and wirewrap pin lengths are not guaranteed to conform to Intel OEM or System packaging.

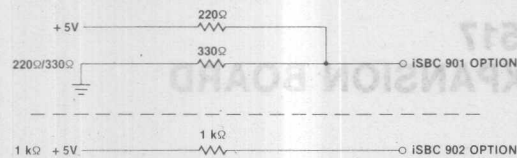
Line Drivers and Terminators

I/O Drivers — The following line drivers and terminators are compatible with all the I/O driver sockets on the iSBC 519:

Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

Note

I = inverting; NI = non-inverting; OC = open-collector.

I/O Terminators — 220 Ω /330 Ω divider or 1 k Ω pullup

Ports 1, 4, and 7 may use any of the drivers or terminators shown above for unidirectional (input or output) port configurations. Either terminator and the following bidirectional drivers and terminators may be used for ports 1, 4, and 7 when these ports are used as bidirectional ports.

Bidirectional Drivers

Driver	Characteristic	Sink Current (mA)
Intel 8216	NI, TS	25
Intel 8226	I, TS	50

Note

I = inverting; NI = non-inverting; TS = three-state.

Terminators (for ports 1, 4, and 7 when used as bidirectional ports)

Supplier	Product Series
CTS	760
Dale	LDP14k-02
Beckman	899-1

Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-state	50
Commands	Tri-state	25

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.50 in. (1.27 cm)

Weight — 14 oz (397.3 gm)

Electrical Characteristics**Average DC Current**

Voltage	Without Termination ¹	With Termination ²
$V_{CC} = +5V \pm 5\%$	$I_{CC} = 1.5A \text{ max}$	3.5A max

Note

1. Does not include power required for optional I/O drivers and I/O terminators.

2. With 18 220 Ω /330 Ω Input terminators installed, all terminator inputs low.

Environmental Characteristics

Operating Temperature — 0°C to +55°C

Reference Manual

9800385B — iSBC 519 hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

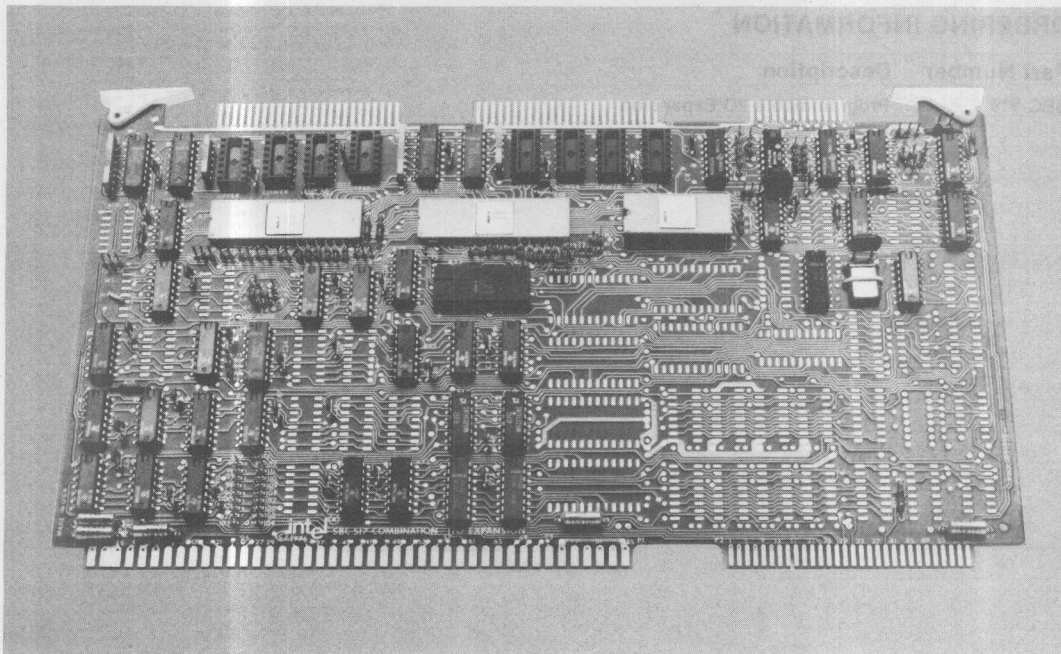
Part Number	Description
SBC 519	Programmable I/O Expansion Board



iSBC 517 COMBINATION I/O EXPANSION BOARD

- 48 programmable I/O lines with sockets for interchangeable line drivers and terminators
- Eight maskable interrupt request lines with a pending interrupt register
- Synchronous/asynchronous communications interface with RS232C drivers and receivers
- 1 ms interval timer

The iSBC 517 Combination I/O Expansion Board is a member of Intel's complete line of iSBC memory and I/O expansion boards. The board interfaces directly with any iSBC single board computer via the system bus to expand serial and parallel I/O capacity. The combination I/O board contains 48 programmable parallel I/O lines. The system software is used to configure the I/O lines to meet a wide variety of system peripheral requirements. The flexibility of the I/O interface is significantly enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. A programmable RS232C communications interface is provided on the iSBC 517. This interface may be programmed by the system software to provide virtually any asynchronous or synchronous serial data transmission technique (including IBM Bi-Sync). A comprehensive RS232C interface to CRTs, RS232C compatible cassettes, and asynchronous and synchronous modems is thus on the board. An on-board register contains the status of eight interrupt request lines which may be interrogated from the system bus, and each interrupt request line is maskable under program control. The iSBC 517 also contains a jumper selectable 1 ms interval timer and interface logic for eight interrupt request lines.



FUNCTIONAL DESCRIPTION

Programming Flexibility

The 48 programmable I/O lines on the iSBC 517 are implemented utilizing two Intel 8255 programmable peripheral interfaces. The system software is used to configure these programmable I/O lines in any of the combinations of unidirectional input/output, and bi-directional ports indicated in Table 1. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 48 programmable I/O lines and signal ground lines are brought out to two 50-pin edge connectors that mate with flat, round, or woven cable. Typical I/O read access time is 280 nanoseconds. Typical I/O read cycle time is 600 nanoseconds.

Communications Interface

The programmable communications interface on the iSBC 517 is provided by an Intel 8251 Universal Synchronous/Asynchronous Receiver/Transmitter (USART). The USART can be programmed by the system software to select the desired asynchronous or synchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity,

and asynchronous serial transmission rate are all under program control. The 8251 provides full duplex, double-buffered transmit and receive capability, and parity, overrun, and framing error detection are all incorporated in the USART. The comprehensive RS232C interface on the board provides a direct interface to RS232C compatible equipment. The RS232C serial data lines and signal ground lines are brought out to a 26-pin edge connector that mates with RS232C compatible flat or round cables.

Interrupt Request Lines

Interrupt requests may originate from eight sources. Four jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface when a byte of information is ready to be transferred to the CPU (i.e., input buffer is full) or a character has been transmitted (i.e., output data buffer is empty). Two jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU (i.e., receive buffer is full) or a character has been transmitted (transmit buffer is empty). These six interrupt request lines are all maskable under program control. Two interrupt request lines may be interfaced directly from user designated peripheral devices via the I/O edge connector. An on-board register contains the status of all eight interrupt request lines, and may be interrogated by the CPU. Each interrupt request line is

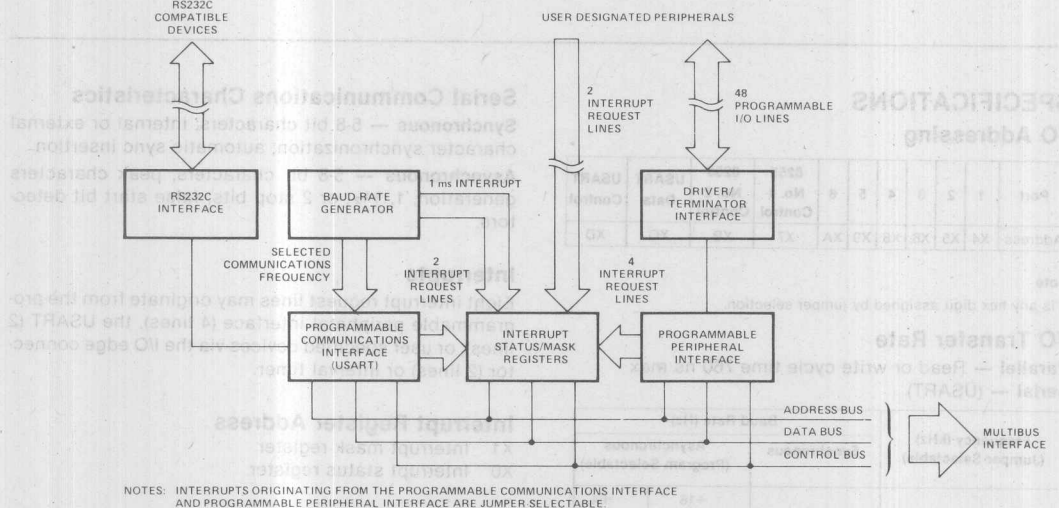


Figure 1. iSBC 517 Combination I/O Expansion Board Block Diagram

Table 1. Input/Output Port Modes of Operation

Ports	Lines (qty)	Mode of Operation					
		Unidirectional				Bidirectional	Control
		Input		Output			
		Unlatched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	4	X		X			X ¹
	4	X		X			X ¹
4	8	X	X	X	X	X	
5	8	X	X	X	X		
6	4	X		X			X ²
	4	X		X			X ²

Notes

1. Part of port 3 must be used as control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output or port 1 is used as a bidirectional port.
2. Part of port 6 must be used as a control port when either port 4 or port 5 are used as a latched and strobed input or a latched and strobed output or port 4 is used as a bidirectional port.

maskable under program control. Routing for the eight interrupt request lines is jumper selectable. They may be ORed to provide a single interrupt request line for the iSBC 80/10B, or they may be individually provided to the system bus for use by other iSBC single board computers.

Interval Timer

Each board contains a jumper selectable 1 ms interval timer. The timer is enabled by jumpering one of the interrupt request lines from the I/O edge connector to a 1 ms interval interrupt request signal originating from the baud rate generator.

SPECIFICATIONS**I/O Addressing**

Port	1	2	3	4	5	6	8255 No. 1 Control	8255 No. 2 Control	USART Data	USART Control
Address	X4	X5	X6	X8	X9	XA	X7	XB	XC	XD

Note

X is any hex digit assigned by jumper selection.

I/O Transfer Rate

Parallel — Read or write cycle time 760 ns max
Serial — (USART)

Frequency (kHz) (Jumper Selectable)	Baud Rate (Hz)			
	Synchronous		Asynchronous (Program Selectable)	
			±16	±64
153.6	—		9600	2400
76.8	—		4800	1200
38.4	38400		2400	600
19.2	19200		1200	300
9.6	9600		600	150
4.8	4800		300	75
6.98	6980		—	110

Serial Communications Characteristics

Synchronous — 5-8 bit characters; internal or external character synchronization; automatic sync insertion.

Asynchronous — 5-8 bit characters; peak characters generation; 1, 1½, or 2 stop bits; false start bit detector.

Interrupts

Eight interrupt request lines may originate from the programmable peripheral interface (4 lines), the USART (2 lines), or user specified devices via the I/O edge connector (2 lines) or interval timer.

Interrupt Register Address

- X1 Interrupt mask register
 X0 Interrupt status register

Note

X is any hex digit assigned by jumper selection.

Timer Interval

- 1.003 ms ± 0.1% when 110 baud rate is selected
 1.042 ms ± 0.1% for all other baud rates

Interfaces

Bus — All signals TTL compatible

Parallel I/O — All signals TTL compatible

Serial I/O — RS232C

Interrupt Requests — All TTL compatible

Connectors

Interface	Pins (qty)	Centers (in.)	Mating Connectors
Bus	86	0.156	CDC VPB01E43A00A1
Parallel I/O	50	0.1	3M 3415-000 or TI H312125
Serial I/O	26	0.1	3M 3462-000 or TI H312113
Auxiliary ¹	60	0.1	AMP PE5-14559 or TI H311130

Note

1. Connector heights and wire-wrap pin lengths are not guaranteed to conform to Intel OEM or system packaging. Auxiliary connector is used for test purposes only.

Line Drivers and Terminators

I/O Drivers — The following line drivers and terminators are compatible with all the I/O driver sockets on the iSBC 517:

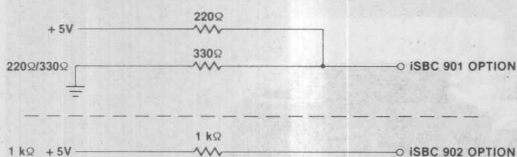
Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

Note

I = inverting; NI = non-inverting; OC = open-collector.

Ports 1 and 4 have 25 mA totem-pole drivers and 1 k Ω terminators.

I/O Terminators — 220 Ω /330 Ω divider or 1 k Ω pullup



Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-state	50
Commands	Tri-state	25

Physical Characteristics

Width — 12.00 in. (30.48 cm)

Height — 6.75 in. (17.15 cm)

Depth — 0.50 in. (1.27 cm)

Weight — 14 oz (397.3 gm)

Electrical Characteristics

Average DC Current

$V_{CC} = +5V \pm 5\%$

$V_{DD} = +12V \pm 5\%$

$V_{AA} = -12V \pm 5\%$

$I_{CC} = 2.4 \text{ mA max}$

$I_{DD} = 40 \text{ mA max}$

$I_{AA} = 60 \text{ mA max}$

Note

Does not include power required for optional I/O drivers and I/O terminators. With eight 220 Ω /330 Ω input terminators installed, all terminator inputs low.

Environmental Characteristics

Operating Temperature — 0°C to +55°C

Reference Manual

9800388B — iSBC 517 hardware Reference Manual (NOT SUPPLIED)¹

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 517	Combination I/O Expansion Board

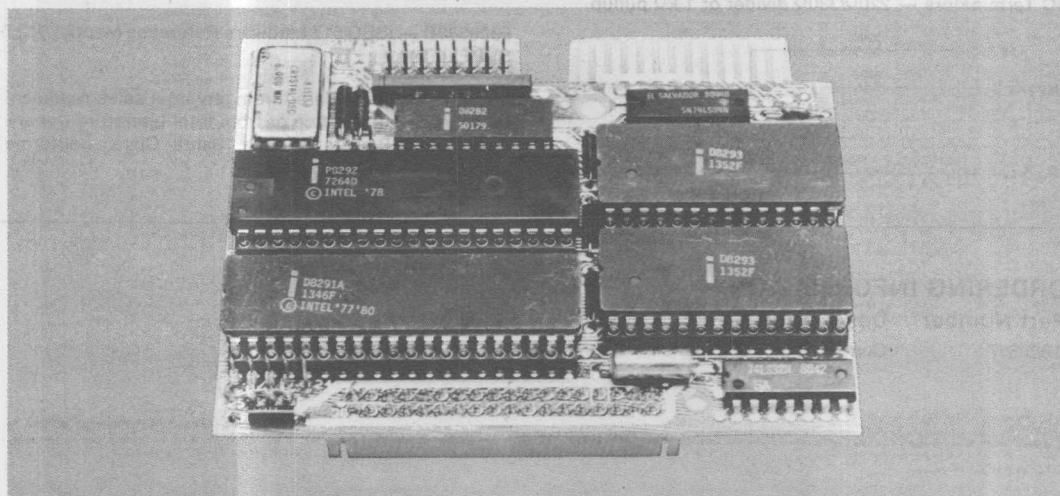


iSBX 488™ GPIB MULTIMODULE BOARD

- **Complete IEEE 488-1978 talker/listener functions including:**
 - Addressing, handshake protocol, service request, serial and parallel polling schemes
- **Complete IEEE 488-1978 controller functions including:**
 - Transfer control, service requests and remote enable
- **Simple read/write programming**
- **Software functions built into VLSI hardware for high performance, low cost and small size**
- **Standard iSBX™ Bus interface for easy connection to Intel iSBC™ boards**
- **IEEE 488-1978 standard electrical interface transceivers**
- **Five volt only operation**

The Intel iSBX 488 GPIB Talker/Listener/Controller Multimodule board provides a standard interface from any Intel iSBC board equipped with an iSBX connector to over 600 instruments and computer peripherals that employ the use of the IEEE 488-1978 standard (General Purpose Interface Bus). The single-wide iSBX 488 Multimodule board implements the complete IEEE 488-1978 Standard Digital Interface for Programmable Instrumentation by taking full advantage of Intel's VLSI technology. The iSBX 488 Multimodule board incorporates the 8291A GPIB Talker/Listener, 8292 GPIB Controller and two 8293 GPIB Transceiver devices on a single low cost 3.7" by 2.85" iSBX Multimodule board. The iSBX 488 board represents a significant step forward in joining microcomputers and instrumentation using industry standards such as the Multibus system bus, iSBX bus and IEEE 488-1978. The high performance iSBX 488 Multimodule board mounts easily on Intel iSBX bus compatible single board computers and provides functions which previously required a board eight times its size.

The iSBX 488 board provides a simple programming interface to the user for easy reading, writing and monitoring of all GPIB functions. The intelligent interface provided by the iSBX 488 board minimizes the impact of the host processor bandwidth.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, MULTIBUS, iRMX, iSBC, iSBX, MULTIMODULE, iCS and iEBC, and the combination of MCS, ICE, iRMX, iSBC, iSBX, iCS or iEBC, and a numerical suffix. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

FUNCTIONAL DESCRIPTION

The iSBX 488 Multimodule board is a single-wide iSBX bus compatible I/O expansion board that provides a complete implementation of the IEEE 488-1978 Standard Digital Interface for Programmable Instrumentation. The iSBX 488 Multimodule board may be configured to be a GPIB controller, talker, listener or talker/listener. The hardware implementation of the iSBX 488 board takes full advantage of Intel's VLSI capability by using the Intel 8292 GPIB controller, 8291A talker/listener and two (2) 8293 bus transceivers. All communication between the host iSBC board and the iSBX 488 Multimodule board is executed via the Intel standard iSBX connector. Many of the functions that previously were performed by user software have been incorporated into VLSI hardware for high performance and simple programming. Both the Intel 8291A GPIB Talker/Listener device and the 8292 device can each communicate independently with the host processor on the iSBC board depending on configuration. Communication from the host iSBC board to either device on the iSBX 488 board is flexible and may be either interrupt or poll driven depending on user requirements. Data transfers to or from the GPIB may be executed by the host processor's I/O Read and I/O Write commands or with DMA

handshaking techniques for very high speed transfers.

GPIB Talker/Listener Capabilities

The Intel 8291A device on the iSBX 488 Multimodule board handles all talker/listener communications between the host iSBC processor board and the GPIB. Its capabilities include data transfer, bus handshake protocol, talker/listener addressing procedures, device clearing and triggering, service requests, and both serial and parallel polling schemes. In executing most procedures the iSBX 488 board does not interrupt the microprocessor on the iSBC processor board unless a byte of data is waiting on input or a byte is sent to an empty output buffer, thus offloading the host CPU of GPIB overhead chores.

SIMPLE PROGRAMMING INTERFACE — The GPIB talker/listener functions can be easily programmed using the high level commands made available by the Intel 8291A on the iSBX 488 Multimodule board. The 8291A device architecture includes eight registers for input and eight registers for output. One each of these read and write registers is used for direct data transfers. The remaining write registers are used by the pro-

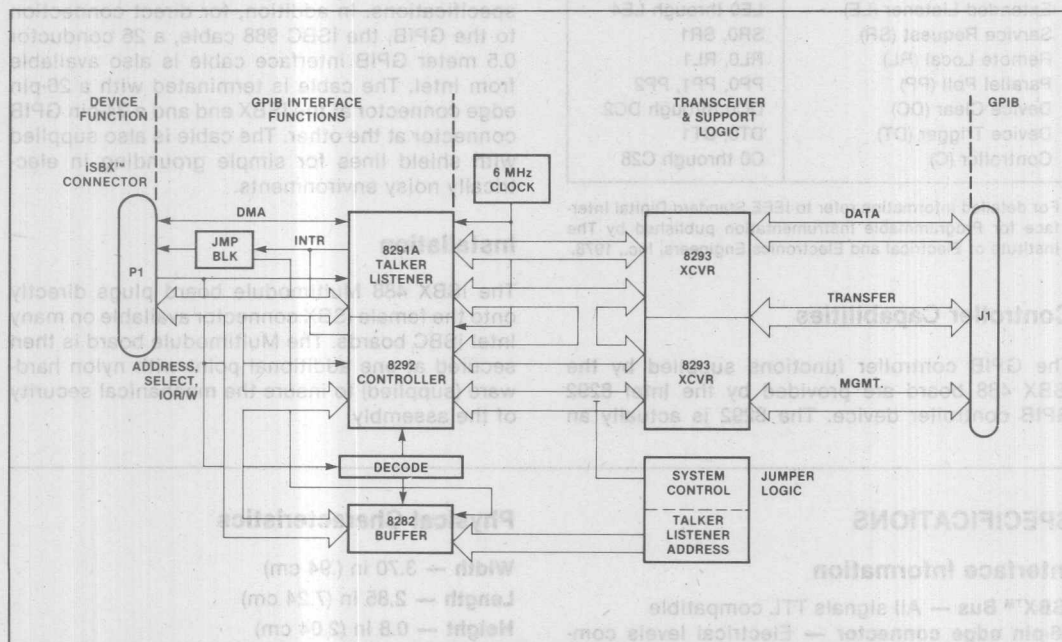


Figure 1. iSBX 488™ Multimodule Board Block Diagram

grammer to control the various interface features of the Intel 8291A device. The remaining read registers provide the user with a monitor of GPIB states, bus conditions and device status.

SOFTWARE FUNCTIONS BUILT INTO VLSI HARDWARE — Additional features that have migrated from discrete logic and software into Intel VLSI include programmable data transfer rate and three addressing modes that allow the iSBX board to be addressed as either a major or a minor talker/listener with primary or secondary addressing. The iSBX 488 Multimodule board can be programmatically configured into almost any bus talker, listener, or talker/listener configuration. Writing software to control these and other iSBX 488 board functions is simply a matter of reading or writing the control registers.

IEEE 488-1978 Functions¹

Function	iSBX 488™ Supported IEEE Subsets
Source Handshake (SH)	SH0, SH1
Acceptor Handshake (AH)	AH0, AH1
Talker (T)	T0 through T8
Extended Talker (TE)	TE0 through TE8
Listener (L)	L0 through L4
Extended Listener (LE)	LE0 through LE4
Service Request (SR)	SR0, SR1
Remote Local (RL)	RL0, RL1
Parallel Poll (PP)	PP0, PP1, PP2
Device Clear (DC)	DC0 through DC2
Device Trigger (DT)	DT0, DT1
Controller (C)	C0 through C28

¹ For detailed information refer to IEEE Standard Digital Interface for Programmable Instrumentation published by The Institute of Electrical and Electronics Engineers, Inc., 1978.

Controller Capabilities

The GPIB controller functions supplied by the iSBX 488 board are provided by the Intel 8292 GPIB controller device. The 8292 is actually an

Intel 8041A eight bit microcomputer that has been preprogrammed to implement all IEEE 488-1978 controller functions. The internal RAM in the 8041A is used as a special purpose register bank for the 8292 GPIB Controller. Just as with the 8291A GPIB Talker/Listener device, these registers are used by the programmer to implement controller monitor, read and write commands on the GPIB.

When configured as a bus controller the iSBX 488 board will respond to Service Requests (SRQ) and will issue Serial Polls. Parallel Polls are also issued to multiple GPIB instrument devices for receiving simultaneous responses. In applications requiring multiple bus controllers, several iSBX 488 boards may each be configured as a controller and pass the active control amongst each other. An iSBX 488 board configured for a System Controller has the capability to send Remote Enable (REN) and Interface Clear (IFC) for initializing the bus to a known state.

GPIB Physical Interface

The iSBX 488 Multimodule board interfaces to the GPIB using two Intel 8293 bidirectional transceivers. The iSBX 488 board meets or exceeds all of the electrical specifications defined in IEEE 488-1978 including the required bus termination specifications. In addition, for direct connection to the GPIB, the iSBC 988 cable, a 26 conductor 0.5 meter GPIB interface cable is also available from Intel. The cable is terminated with a 26-pin edge connector at the iSBX end and a 24-pin GPIB connector at the other. The cable is also supplied with shield lines for simple grounding in electrically noisy environments.

Installation

The iSBX 488 Multimodule board plugs directly onto the female iSBX connector available on many Intel iSBC boards. The Multimodule board is then secured at one additional point with nylon hardware (supplied) to insure the mechanical security of the assembly.

SPECIFICATIONS

Interface Information

iSBX™ Bus — All signals TTL compatible

26-pin edge connector — Electrical levels compatible with IEEE 488-1978.

Physical Characteristics

Width — 3.70 in (.94 cm)

Length — 2.85 in (7.24 cm)

Height — 0.8 in (2.04 cm)

Weight — 3.1 oz (87.8 gm)

GPIO Data Rate*

300K bytes/sec transfer rate with DMA host iSBC board

50K bytes/sec transfer rate using programmed I/O

730 nsec Data Accept Time

* Data rates are iSBX board maximum. Data rates will vary and can be slower depending on host iSBC board and user software driver.

Electrical Characteristics**DC power requirements —**

V_{CC} = +5 Vdc ±5%

I_{CC} = 600 milliamps maximum

GPIO Electrical and Mechanical Specifications

Conforms to IEEE 488-1978 standard electrical levels and mechanical connector standard when purchased with the iSBC 988 GPIO cable.

Environmental Characteristics

Operating Temperature — 0° to 60°C (32° to 140°F)

Relative Humidity — Up to 90% R.H. without condensation.

Reference Manual

143154-001 — iSBX 488 GPIO Multimodule Board Hardware Reference Manual (not supplied).

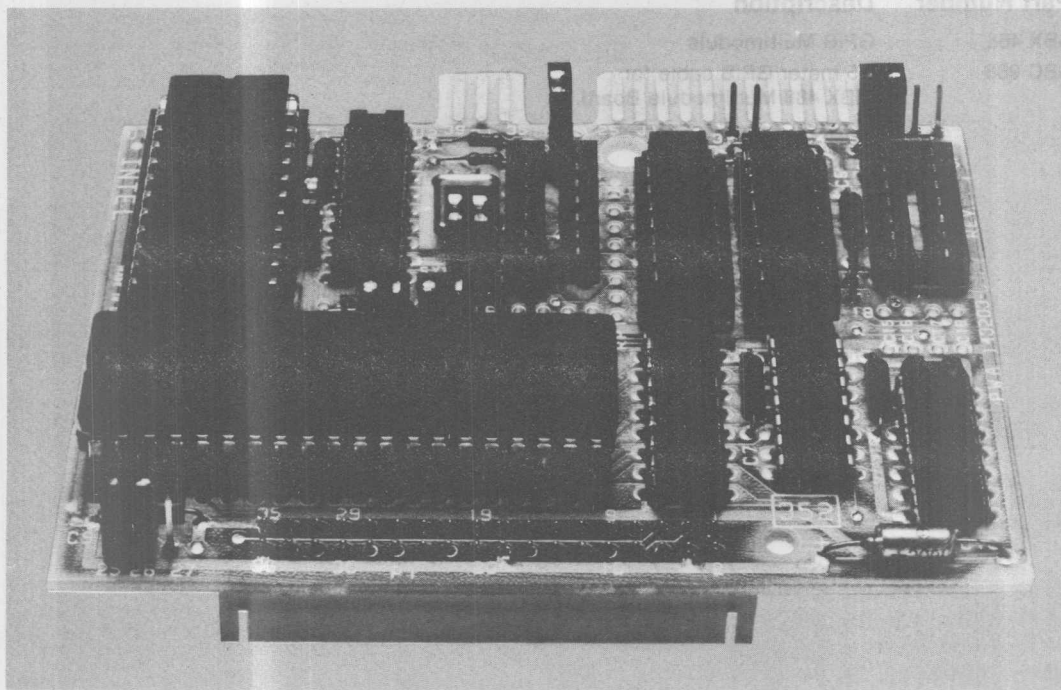
ORDERING INFORMATION

Part Number	Description
SBX 488	GPIO Multimodule
SBC 988	0.5 meter GPIO cable for iSBX 488 Multimodule Board

iSBX™ 352 BIT SERIAL COMMUNICATIONS MULTIMODULE™ BOARD

- Provides an HDLC/SDLC half/full-duplex communications channel for iSBX™ bus compatible micro-computers
- Supports RS232C (including modem support) or RS449/422A interface
- Single +5V when configured for RS449/422A interface
- Software programmable baud rate generation up to 64K baud synchronous and 9.6K baud self-clocking
- Supports synchronous or self-clocking NRZI point-to-point, multidrop and self-clocking NRZI SDLC loop data link interfaces

The Intel iSBX 352 Bit Serial Communications MULTIMODULE board offers incremental on-board I/O expansion support for ISO/CCITT's HDLC or IBM's SDLC communication. Plugging directly into any iSBX bus compatible host board, the iSBX 352 module provides one RS232C or RS449/422A programmable bit serial communications channel with software selectable baud rates (up to 64K baud for half-duplex synchronous operations). Data link interfaces supported are: synchronous point-to-point, multidrop and SDLC loop. The phase lock loop feature provides NRZI self-clocking 9.6K baud operation.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

FUNCTIONAL DESCRIPTION

Communications Interface

The iSBX 352 module uses the Intel 8273 Programmable HDLC/SDLC Protocol Controller. The iSBX 352 module provides one bit-serial communications channel for iSBX bus compatible host microcomputers. (See Figure 1.) An iSBC microcomputer or MULTIBUS-based application is easily connected to an HDLC/SDLC point-to-point, multidrop, or an SDLC loop configuration.

The High-Level Data Link Control (HDLC) is the International Standards Organization (ISO) standard discipline used to implement X.25 packet switching communications. The Synchronous Data Link Control (SDLC) is an IBM communication protocol used to implement the System Network Architecture (SNA). Both protocols, HDLC and SDLC, are bit oriented, code independent, and support full-duplex operations.

Data Link Interface

The control lines, serial data lines and signal ground lines are brought out to the double edge connector of the iSBX 352 module and are configurable for RS232C or RS449/422A interface (see Figure 2).

Addressing an iSBX 352 board by using a port address, the program performs the 8-bit data transfer required, using buffered or non-buffered transmit/receive and abort sequences.

Serial data transfer control is provided by the 8273 controller of the iSBX 352 module which interfaces the parallel iSBX bus to the serial channel. During a transmit sequence, the iSBX 352 module accepts data and commands from the iSBX bus interface, translates and formats the data into HDLC/SDLC protocol formats, provides the proper RS232C or RS422A interface control signals, and passes data onto the serial channel. The receive operation is the inverse of the previous sequence.

Data Link Configurations

The supported data link configurations are shown in Table 1. The following example configurations provide an overview and a figure for five typical data link configurations:

Table 1. iSBX™ 352 Supported Configurations

Connection	Synchronous		Asynchronous	
	Modem	Direct	Modem*	Direct
point-to-point	X	X	X	X
multidrop	X	X	X	X
loop	NA	NA	X	X

* Modem should not respond to a break.

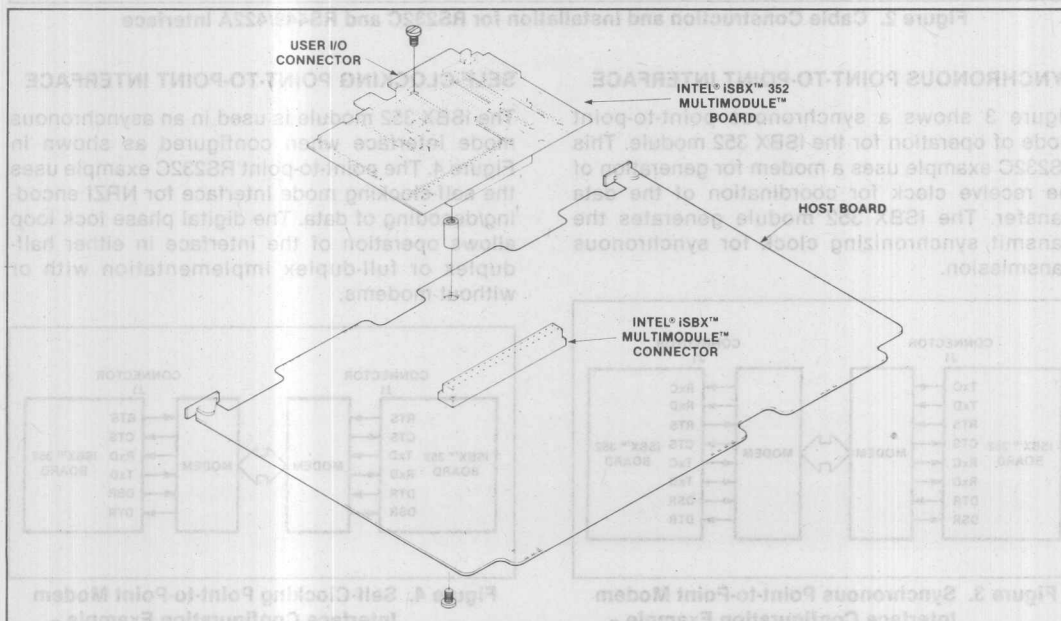


Figure 1. Installation of iSBX™ 352 MULTIMODULE™ Board on a Host Board

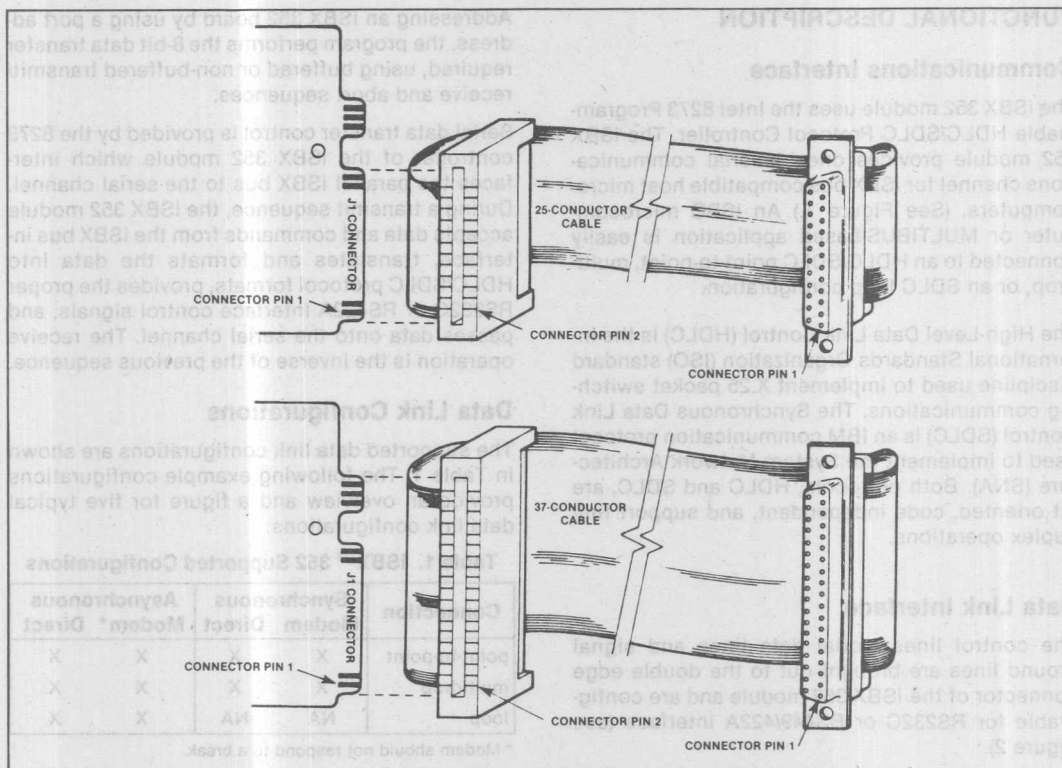


Figure 2. Cable Construction and Installation for RS232C and RS449/422A Interface

SYNCHRONOUS POINT-TO-POINT INTERFACE

Figure 3 shows a synchronous point-to-point mode of operation for the iSBX 352 module. This RS232C example uses a modem for generation of the receive clock for coordination of the data transfer. The iSBX 352 module generates the transmit synchronizing clock for synchronous transmission.

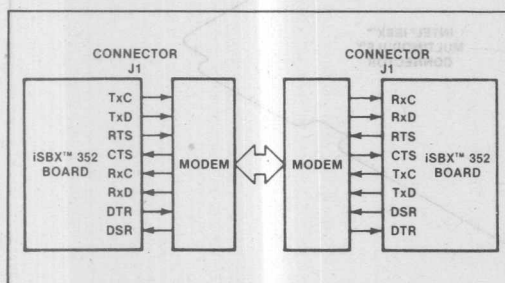


Figure 3. Synchronous Point-to-Point Modem Interface Configuration Example - RS232C

SELF-CLOCKING POINT-TO-POINT INTERFACE

The iSBX 352 module is used in an asynchronous mode interface when configured as shown in Figure 4. The point-to-point RS232C example uses the self-clocking mode interface for NRZI encoding/decoding of data. The digital phase lock loop allows operation of the interface in either half-duplex or full-duplex implementation with or without modems.

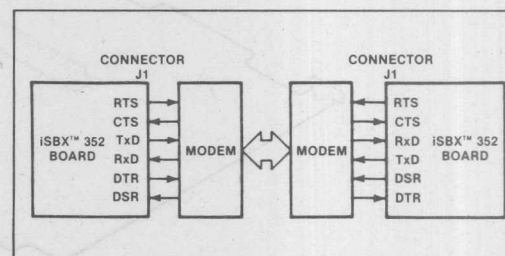


Figure 4. Self-Clocking Point-to-Point Modem Interface Configuration Example - RS232C

SYNCHRONOUS MULTIDROP

The iSBX 352 MULTIMODULE is used in both a master and a slave mode in the RS449/422A example shown in Figure 5. This synchronous multidrop application is effective for high-speed data transfers between slave stations and a central master station.

ASYNCHRONOUS SELF-CLOCKING MULTIDROP

The iSBX 352 MULTIMODULE example in Figure 6 shows a master and multiple slaves in a multidrop

configuration. This self-clocking example uses the 8273 digital phase lock loop and NRZI data encoding.

SDLC Loop

The SDLC self-clocking loop configuration shown in Figure 7 permits longer networks since each secondary slave station is a repeater set in one-bit-delay mode. The data sent out by the primary station (the loop controller) are relayed bit-for-bit through each secondary station and finally back to the master station.

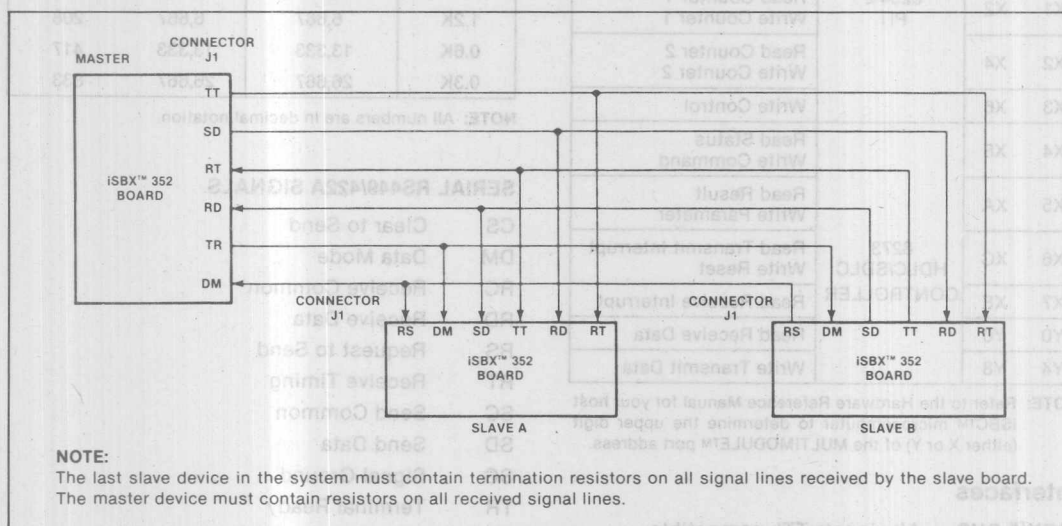


Figure 5. Synchronous Multidrop Network Configuration Example - RS422A

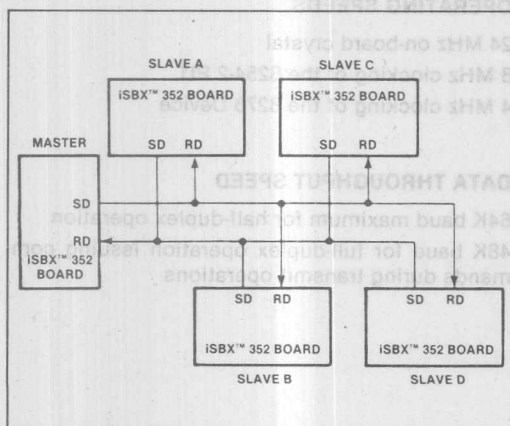


Figure 6. Self-Clocking Multidrop Configuration Example - RS422A

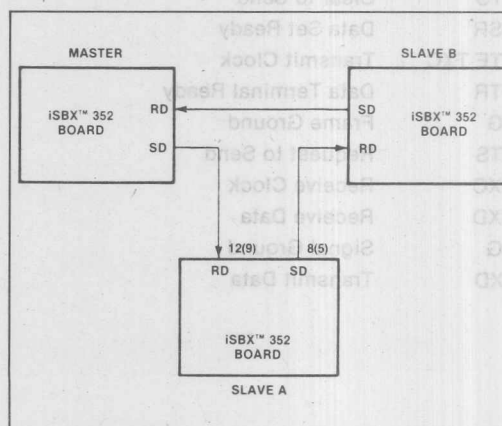


Figure 7. Self-Clocking SDLC Loop Network Configuration Example

SPECIFICATIONS

Data Size

8 Bits

I/O Port Addresses

Port Address	Device Selected	Function Performed
8-bit 16-bit		
X0 X0	8254-2 PIT	Read Counter 0 Write Counter 0
X1 X2		Read Counter 1 Write Counter 1
X2 X4		Read Counter 2 Write Counter 2
X3 X6		Write Control
X4 X8	8273 HDLC/SDLC CONTROLLER	Read Status Write Command
X5 XA		Read Result Write Parameter
X6 XC		Read Transmit Interrupt Write Reset
X7 XE		Read Receive Interrupt
Y0 Y0		Read Receive Data
Y4 Y8		Write Transmit Data

NOTE: Refer to the Hardware Reference Manual for your host ISBC™ microcomputer to determine the upper digit (either X or Y) of the MULTIMODULE™ port address.

Interfaces

ISBX™ BUS — All signals TTL compatible

SERIAL RS232C SIGNALS

CTS	Clear to Send
DSR	Data Set Ready
DTE TXC	Transmit Clock
DTR	Data Terminal Ready
FG	Frame Ground
RTS	Request to Send
RXC	Receive Clock
RXD	Receive Data
SG	Signal Ground
TXD	Transmit Data

RATE GENERATOR FREQUENCIES

Baud Rate bits/sec	8254-2 Divide Count	
	Synchronous	Self-Clocking
64K	125	TX Clock 32X Clock
56K	143	— —
48K	167	— —
19.2K	417	— —
9.6K	833	833 26
4.8K	1,667	1,667 52
2.4K	3,333	3,333 104
1.2K	6,667	6,667 208
0.6K	13,333	13,333 417
0.3K	26,667	26,667 833

NOTE: All numbers are in decimal notation.

SERIAL RS449/422A SIGNALS

CS	Clear to Send
DM	Data Mode
RC	Receive Common
RD	Receive Data
RS	Request to Send
RT	Receive Timing
SC	Send Common
SD	Send Data
SG	Signal Ground
TR	Terminal Ready
TT	Terminal Timing

OPERATING SPEEDS

- 24 MHz on-board crystal
- 8 MHz clocking of the 8254-2 PIT
- 4 MHz clocking of the 8273 Device

DATA THROUGHPUT SPEED

- 64K baud maximum for half-duplex operation
- 48K baud for full-duplex operation issuing commands during transmit operations

SERIAL INTERFACE CONNECTORS

Configuration	Mode ²	MULTIMODULE™ Edge Connector	Cable	Connector
RS232C	DTE	26-pin ⁵ , 3M-3462-0001	3M ³ -3349/25	25-pin ⁷ , 3M-3482-1000
RS232C	DCE	26-pin ⁵ , 3M-3462-0001	3M ³ -3349/25	25-pin ⁷ , 3M-3483-1000
RS449	DTE	40-pin ⁶ , 3M-3464-0001	3M ⁴ -3349/37	37-pin ¹ , 3M-3502-1000
RS449	DCE	40-pin ⁶ , 3M-3464-0001	3M ⁴ -3349/37	37-pin ¹ , 3M-3503-1000

NOTES:

1. Cable housing 3M-3485-4000 may be used with the connector.
2. DTE - Data Terminal Equipment mode (male connector); DCE - Data Set Equipment mode (female connector).
3. Cable is tapered at one end to fit the 3M-3462 connector.
4. Cable is tapered to fit 3M-3464 connector.
5. Pin 26 of the edge connector is not connected to the flat cable.
6. Pins 38, 39, and 40 of the edge connector are not connected to the flat cable.
7. May be used with the cable housing 3M-3485-1000.

Electrical Characteristics

DC POWER REQUIREMENTS

Interface	Voltage	Current (max)	Total Power
RS 232C	+ 5 ± 0.25V	595 mA	3.8 watts
	- 12 ± 0.6V	30 mA	
	+ 12 ± 0.6V	30 mA	
RS 449/422A	+ 5 ± 0.25V	775 mA	4.1 watts

Environmental Characteristics

Temperature — 0 - 55°C, free moving air across base board and MULTIMODULE board

Humidity — to 90%, without condensation

Physical Characteristics

Width — 7.27 cm (2.85 inches)

Length — 9.40 cm (3.70 inches)

Height — 1.40 cm (0.56 inches)

Weight — 72 gm (2.53 ounces)

Reference Manual (Not Supplied)

143983 — ISBX 352 Bit Serial Communications MULTIMODULE Board Hardware Reference Manual.

Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Ave., Santa Clara, California 95051.

ORDERING INFORMATION

Part Number Description

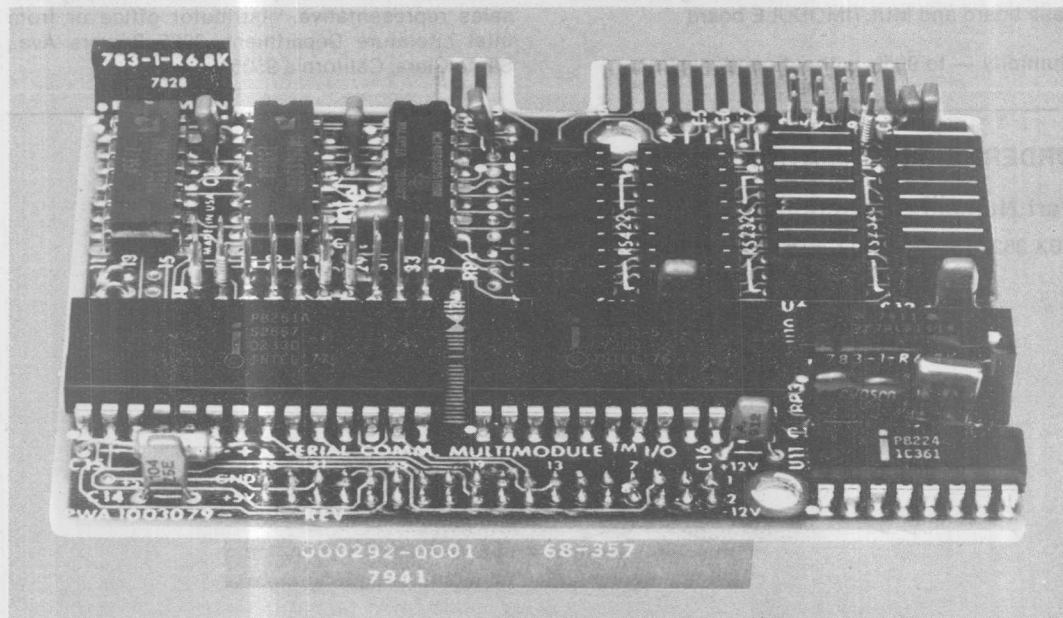
SBX 352 HDLC/SDLC Serial I/O
MULTIMODULE Board

iSBX 351

SERIAL I/O MULTIMODULE BOARD

- iSBX bus compatible I/O expansion
- Programmable synchronous/asynchronous communications channel with RS232C or RS449/422 interface
- Software programmable baud rate generator
- Two programmable 16-bit BCD or binary timers/event counters
- Four jumper selectable interrupt request sources
- Accessed as I/O port locations
- Low power requirements
- Single +5V when configured for RS449/422 interface
- iSBX bus on-board expansion eliminates MULTIBUS system bus latency and increases system throughput

The Intel® iSBX 351 Serial I/O MULTIMODULE board is a member of Intel's new line of iSBX bus compatible MULTIMODULE products. The iSBX MULTIMODULE board plugs directly into any iSBX bus compatible host board offering incremental on-board I/O expansion. The iSBX 351 module provides one RS232C or RS449/422 programmable synchronous/asynchronous communications channel with software selectable baud rates. Two general purpose programmable 16-bit BCD or binary timers/event counters are available to the host board to generate accurate time intervals under software control. The iSBX board is closely coupled to the host board through the iSBX bus, and as such, offers maximum on-board performance and frees MULTIBUS system traffic for other system resources. In addition, incremental power dissipation is minimum requiring only 3.0 watts (assumes RS232C interface).



FUNCTIONAL DESCRIPTION

Communications Interface

The iSBX 351 module uses the Intel® 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) providing one programmable communications channel. The USART can be programmed by the system software to individually select the desired asynchronous or synchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e. synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun,

and framing error detection are all incorporated in the USART. The command lines, serial data lines, and signal ground lines are brought out to a double edge connector configurable for either an RS232C or RS449/422 interface (see Figure 3). In addition, the iSBX 351 module is jumper configurable for either point-to-point or multidrop network connection.

16-Bit Interval Timers

The iSBX 351 module uses an Intel 8253 Programmable Interval Timer (PIT) providing 3 fully programmable and independent BCD and binary 16-bit

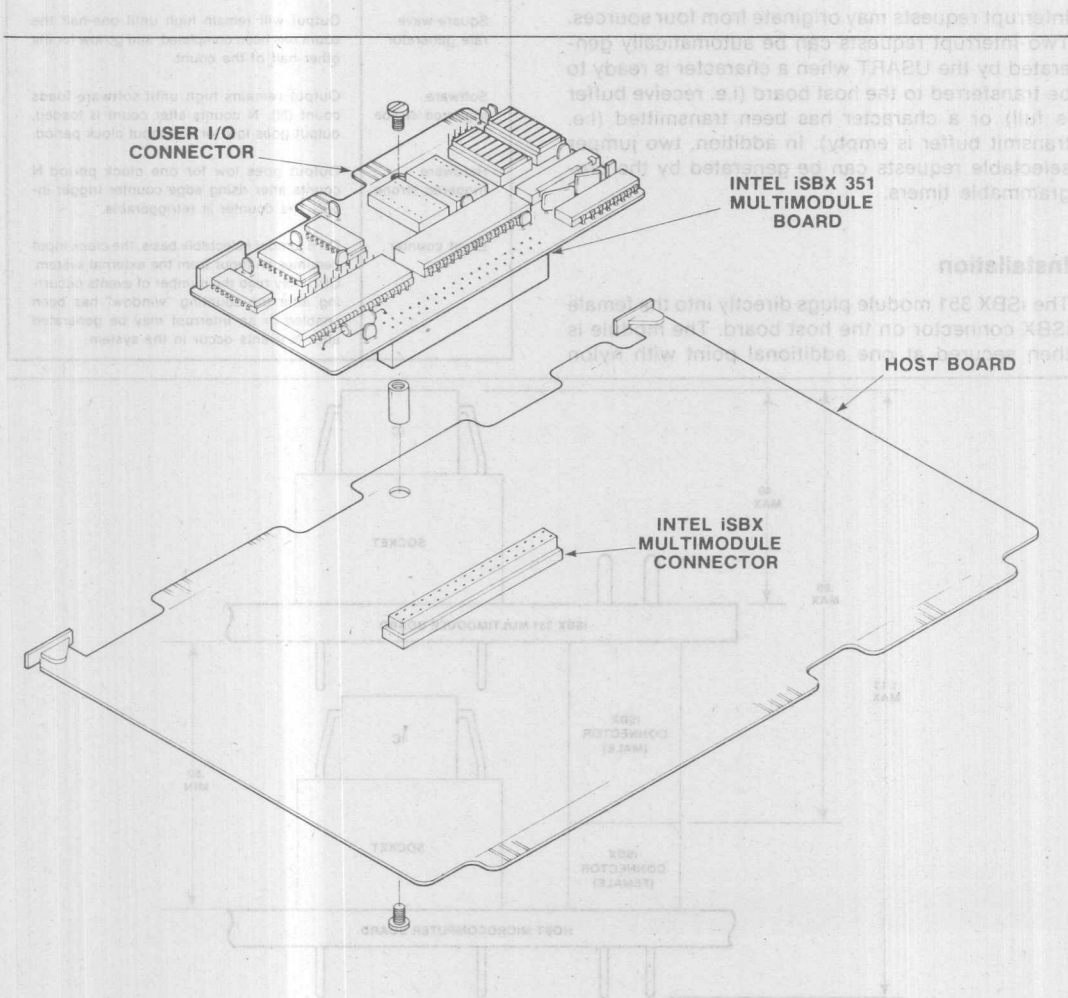


Figure 1. Installation of iSBX 351 Module on a Host Board

interval timers. One timer is available to the system designer to generate baud rates for the USART under software control. Routing for the outputs from the other two counters is jumper selectable to the host board. In utilizing the iSBX 351 module, the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given baud rate or time delay is needed, software commands the programmable timers to select the desired function. The functions of the timers are shown in Table 1. The contents of each counter may be read at any time during system operation.

Interrupt Request Lines

Interrupt requests may originate from four sources. Two interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the host board (i.e. receive buffer is full) or a character has been transmitted (i.e. transmit buffer is empty). In addition, two jumper selectable requests can be generated by the programmable timers.

Installation

The iSBX 351 module plugs directly into the female iSBX connector on the host board. The module is then secured at one additional point with nylon

hardware to insure the mechanical security of the assembly (see Figures 1 and 2).

Table 1. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached, an interrupt request is generated. This function is useful for generation of real-time clocks.
Programmable one-shot	Output goes low upon receipt of an external trigger edge and returns high when terminal count is reached. This function is retriggerable.
Rate generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-wave rate generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software triggered strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware triggered strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counting "window" has been enabled or an interrupt may be generated after N events occur in the system.

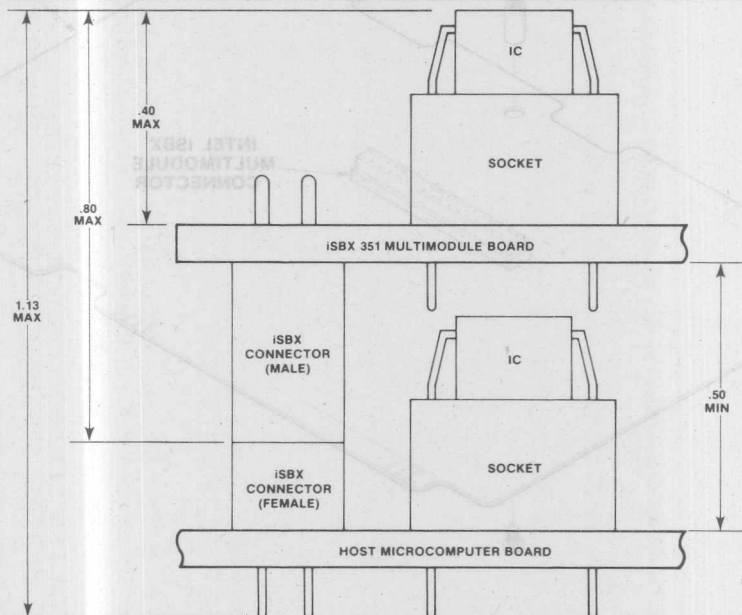


Figure 2. Mounting Clearances (inches)

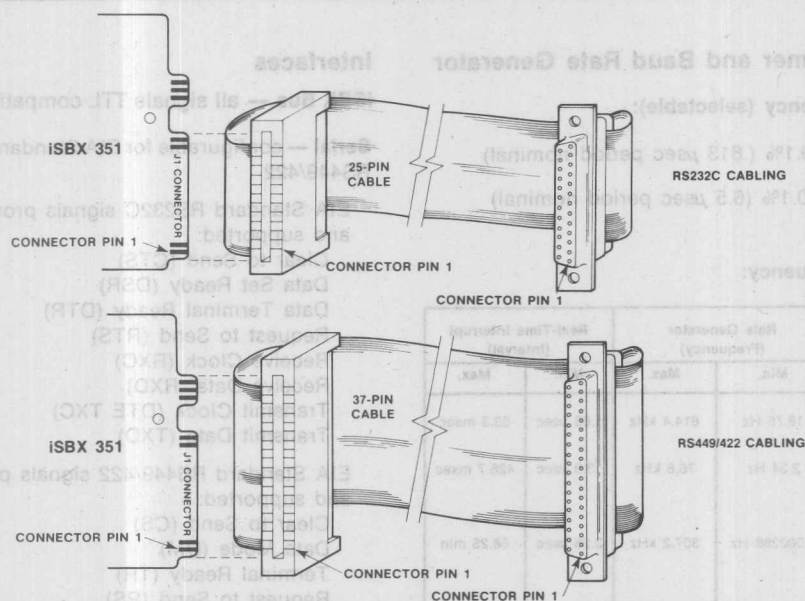


Figure 3. Cable Construction and Installation for RS232C and RS449/422 Interface

SPECIFICATIONS

Word Size

Data — 8 bits

I/O Addressing

I/O Address	Chip Select	Function
X0, X2, X4, or X6	8251A USART	Write: Data Read: Data
X1, X3, X5, or X7		Write: Mode or Command Read: Status
X8 or XC	8253 PIT	Write: Counter 0 (Load Count ÷ N) Read: Counter 0
X9 or XD		Write: Counter 1 (Load Count ÷ N) Read: Counter 1
XA or XE		Write: Counter 2 (Load Count ÷ N) Read: Counter 2
XB or XF		Write: Control Read: None

NOTE: The first digit of each port I/O address is listed as "X" since it will change depending on the type of host iSBC microcomputer used. Refer to the Hardware Reference Manual for your host iSBC microcomputer to determine the first digit of the I/O address.

Access Time

Read — 250 nsec max

Write — 300 nsec max

Note

Actual transfer speed is dependent upon the cycle time of the host microcomputer.

Serial Communications

Synchronous — 5 - 8-bit characters; internal character synchronization; automatic sync insertion; even, odd or no parity generation/detection.

Asynchronous — 5 - 8-bit characters; break character generation and detection; 1, 1½, or 2 stop bits; false start bit detection; even, odd or no parity generation/detection.

Sample Baud Rate:

8253 PIT Frequency ¹ (kHz, Software Selectable)	8251 USART Baud Rate (Hz) ²	
	Synchronous	Asynchronous
307.2	—	÷ 16 ÷ 64 19200 4800
153.6	—	9600 2400
76.8	—	4800 1200
38.4	38400	2400 600
19.2	19200	1200 300
9.6	9600	600 150
4.8	4800	300 75
2.4	2400	150 —
1.76	1760	110 —

NOTES: 1. Frequency selected by I/O writes of appropriate 16-bit frequency factor to Baud Rate Register.

2. Baud rates shown here are only a sample subset of possible software-programmable rates available. Any frequency from 18.75 Hz to 614.4 kHz may be generated utilizing on-board crystal oscillator and 16-bit Programmable Interval Timer (used here as frequency divider).

Interval Timer and Baud Rate Generator

Input Frequency (selectable):

1.23 MHz $\pm 0.1\%$ (.813 μ sec period nominal)

153.6 kHz $\pm 0.1\%$ (6.5 μ sec period nominal)

Output Frequency:

	Rate Generator (Frequency)		Real-Time Interrupt (Interval)	
	Min.	Max.	Min.	Max.
Single Timer ¹	18.75 Hz	614.4 kHz	1.63 μ sec	53.3 msec
Single Timer ²	2.34 Hz	76.8 kHz	13.0 μ sec	426.7 msec
Dual Timer ³ (Counters 0 and 1 in series)	0.000286 Hz	307.2 kHz	3.26 μ sec	58.25 min
Dual Timer ⁴ (Counters 0 and 1 in series)	0.0000358 Hz	38.4 kHz	26.0 μ sec	7.77 hrs

- NOTES:** 1. Assuming 1.23 MHz clock input.
 2. Assuming 153.6 kHz clock input.
 3. Assuming Counter 0 has 1.23 MHz clock input.
 4. Assuming Counter 0 has 153.6 kHz clock input.

Interrupts

Interrupt requests may originate from the USART (2) or the programmable timer (2).

Serial Interface Connectors

Configuration	Mode ²	MULTIMODULE Edge Connector	Cable	Connector ⁸
RS232C	DTE	26-pin ⁵ , 3M-3462-0001	3M ³ -3349/25	25-pin ⁷ , 3M-3482-1000
RS232C	DCE	26-pin ⁵ , 3M-3462-0001	3M ³ -3349/25	25-pin ⁷ , 3M-3483-1000
RS449	DTE	40-pin ⁶ , 3M-3464-0001	3M ⁴ -3349/37	37-pin ¹ , 3M-3502-1000
RS449	DCE	40-pin ⁶ , 3M-3464-0001	3M ⁴ -3349/37	37-pin ¹ , 3M-3503-1000

- NOTES:** 1. Cable housing 3M-3485-4000 may be used with the connector.
 2. DTE — Data Terminal mode (male connector), DCE — Data Set mode (female connector).
 3. Cable is tapered at one end to fit the 3M-3462 connector.
 4. Cable is tapered to fit 3M-3464 connector.
 5. Pin 26 of the edge connector is not connected to the flat cable.
 6. Pins 37, 39, and 40 of the edge connector are not connected to the flat cable.
 7. May be used with cable housing 3M-3485-1000.
 8. Connectors compatible with those listed may also be used.

Interfaces

ISBX Bus — all signals TTL compatible.

Serial — configurable for EIA Standards RS232C or RS449/422

EIA Standard RS232C signals provided and supported:

- Clear to Send (CTS)
- Data Set Ready (DSR)
- Data Terminal Ready (DTR)
- Request to Send (RTS)
- Receive Clock (RXC)
- Receive Data (RXD)
- Transmit Clock (DTE TXC)
- Transmit Data (TXD)

EIA Standard RS449/422 signals provided and supported:

- Clear to Send (CS)
- Data Mode (DM)
- Terminal Ready (TR)
- Request to Send (RS)
- Receive Timing (RT)
- Receive Data (RD)
- Terminal Timing (TT)
- Send Data (SD)

Physical Characteristics

- Width** — 7.24 cm (2.85 inches)
- Length** — 9.40 cm (3.70 inches)
- Height*** — 2.04 cm (0.80 inches)
 - ISBX 351 Board
 - 2.86 cm (1.13 inches)
 - ISBX 351 Board and Host Board
- Weight** — 51 grams (1.79 ounces)

* (See Figure 2)

Electrical Characteristics**DC Power Requirements**

Mode	Voltage	Amps (Max.)
RS232C	+5V ± 0.25 V	460 mA
	+12V ± 0.6 V	30 mA
	-12V ± 0.6 V	30 mA
RS449/422	+5V ± 0.25 V	530 mA

Environmental Characteristics

Temperature — 0 - 55°C, free moving air across the base board and MULTIMODULE board.

Reference Manual

9803190-01 — iSBX 351 Serial I/O MULTIMODULE Manual (NOT SUPPLIED)

Reference Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Ave., Santa Clara, California, 95051.

ORDERING INFORMATION

Part Number	Description
SBX 351	Serial I/O MULTIMODULE Board

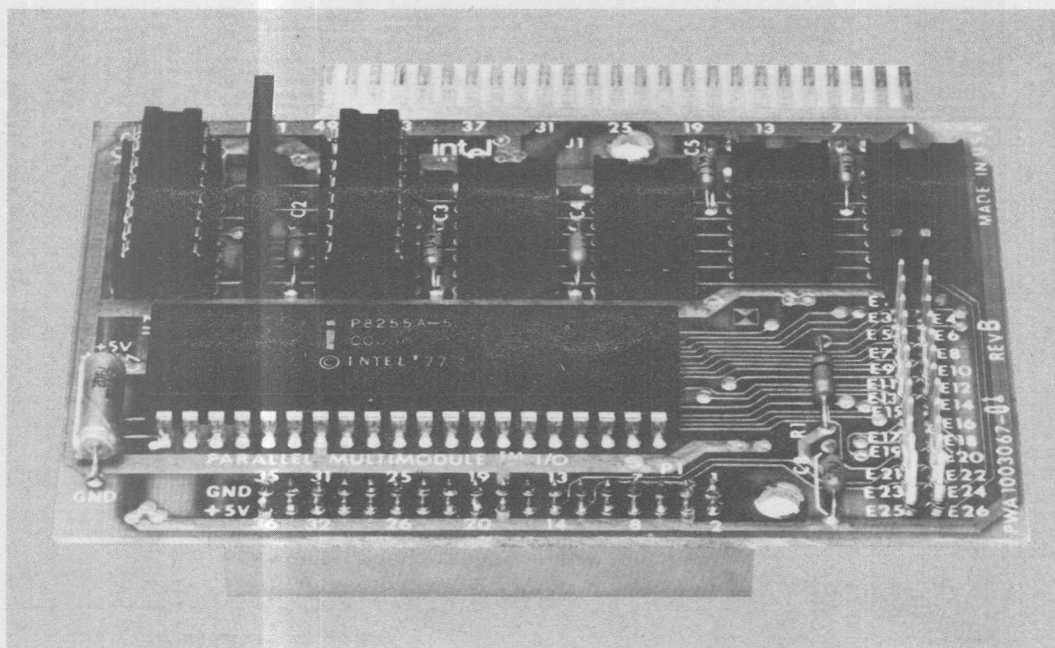


iSBX 351

iSBX 350 PARALLEL I/O MULTIMODULE BOARD

- iSBX bus compatible I/O expansion
- 24 programmable I/O lines with sockets for interchangeable line drivers and terminators
- Three jumper selectable interrupt request sources
- Accessed as I/O port locations
- Single +5V low power requirement
- iSBX bus on-board expansion eliminates MULTIBUS system bus latency and increases system throughput

The Intel® iSBX 350 Parallel I/O MULTIMODULE Board is a member of Intel's new line of iSBX bus compatible MULTIMODULE products. The iSBX MULTIMODULE board plugs directly into any iSBX bus compatible host board offering incremental on-board expansion. The iSBX 350 module provides 24 programmable I/O lines with sockets for interchangeable line drivers and terminators. The iSBX board is closely coupled to the host board through the iSBX bus, and as such, offers maximum on-board performance and frees MULTIBUS system traffic for other system resources. In addition, incremental power dissipation is minimal requiring only 1.6 watts (not including optional driver/terminators).



FUNCTIONAL DESCRIPTION

Programmable Interface

The iSBX 350 module uses an Intel® 8255A-5 Programmable Peripheral Interface (PPI) providing 24 parallel I/O lines. The base-board system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports indicated in Table 1. Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and driver/termination characteristics for each application. In addition, inverting bidirectional bus drivers (8226) are provided on sockets to allow convenient optional replacement to non-inverting drivers (8216). The 24 programmable I/O lines, signal ground, and +5

volt power (jumper configurable) are brought to a 50-pin edge connector that mates with flat, woven, or round cable.

Interrupt Request Generation

Interrupt requests may originate from three jumper selectable sources. Two interrupt requests can be automatically generated by the PPI when a byte of information is ready to be transferred to the base board CPU (i.e., input buffer is full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). A third interrupt source may originate directly from the user I/O interface (J1 connector).

Installation

The iSBX 350 module plugs directly into the female iSBX connector on the host board. The module is then secured at one additional point with nylon hardware to insure the mechanical security of the assembly (see Figure 1 and Figure 2).

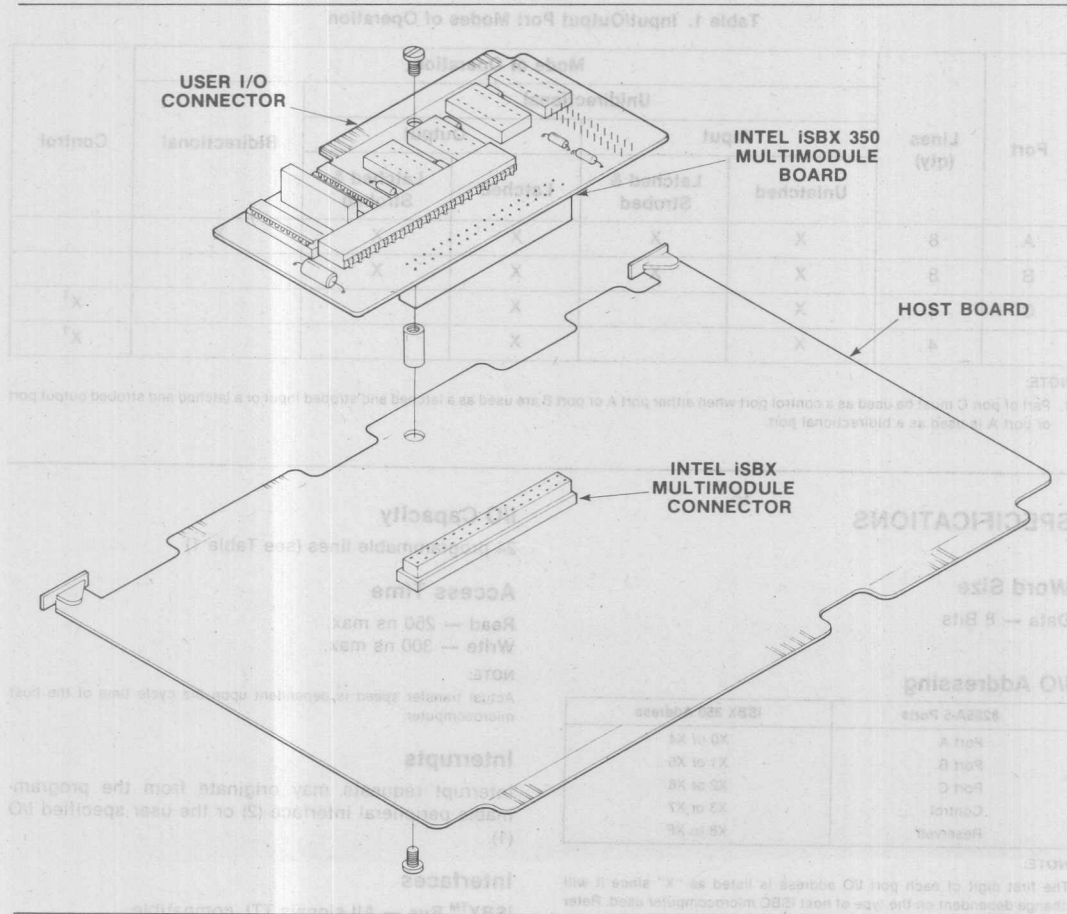


Figure 1. Installation of iSBX 350 Module on a Host Board

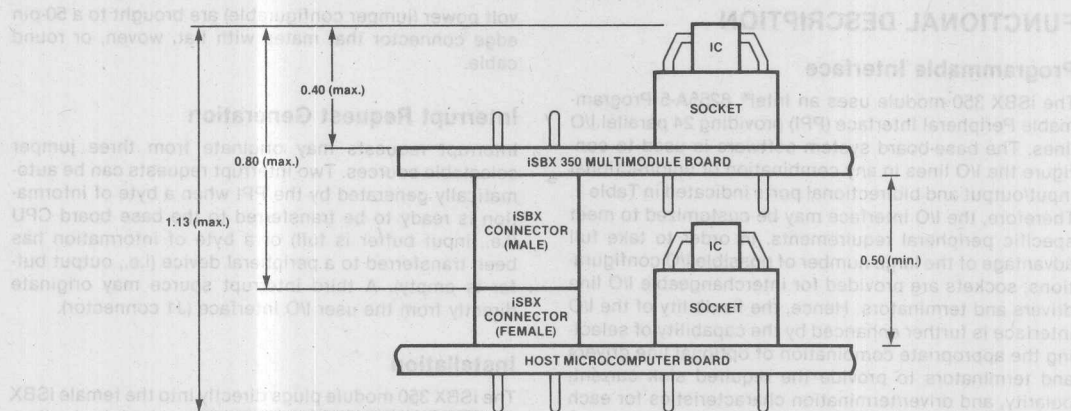


Figure 2. Mounting Clearances (inches)

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation				Control	
		Unidirectional					
		Input		Output			Bidirectional
		Unlatched	Latched & Strobed	Latched	Latched & Strobed		
A	8	X	X	X	X		
B	8	X	X	X	X		
C	4	X		X		X ¹	
	4	X		X		X ¹	

NOTE:

1. Part of port C must be used as a control port when either port A or port B are used as a latched and strobed input or a latched and strobed output port or port A is used as a bidirectional port.

SPECIFICATIONS

Word Size

Data — 8 Bits

I/O Addressing

8255A-5 Ports	ISBX 350 Address
Port A	X0 or X4
Port B	X1 or X5
Port C	X2 or X6
Control	X3 or X7
Reserved	X8 to XF

NOTE:

The first digit of each port I/O address is listed as "X" since it will change dependent on the type of host ISBC microcomputer used. Refer to the Hardware Reference Manual for your host ISBC microcomputer to determine the first digit of the port address.

I/O Capacity

24 programmable lines (see Table 1)

Access Time

Read — 250 ns max.

Write — 300 ns max.

NOTE:

Actual transfer speed is dependent upon the cycle time of the host microcomputer.

Interrupts

Interrupt requests may originate from the programmable peripheral interface (2) or the user specified I/O (1).

Interfaces

ISBX™ Bus — All signals TTL compatible

Parallel I/O — All signals TTL compatible

ISBX 350

Parallel Interface Connectors

Interface	No. of Pairs/Pins	Centers (in.)	Connector Type	Vendor	Vendor Part No.
Parallel I/O Connector	25/50	0.1	Female	3M	3415-0001 with Ears
Parallel I/O Connector	25/50	0.1	Female, Soldered	GTE Sylvania	6AD01251A1DD

Note: Connector compatible with those listed may also be used.

Line Drivers and Terminators

I/O Drivers — The following line drivers and terminators are all compatible with the I/O driver sockets on the ISBX 350.

Driver	Characteristic	Sink Current (mA)
7438	I, OC	48
7437	I	48
7432	NI	16
7426	I, OC	16
7409	NI, OC	16
7408	NI	16
7403	I, OC	16
7400	I	16

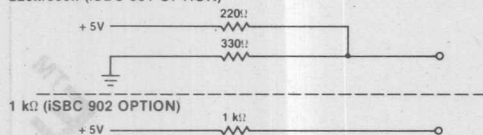
Note:

I = Inverting, NI = Non-Inverting, OC = Open Collector

Port 1 has 25 mA totem pole drivers and 1 k Ω terminators.

I/O Terminators — 220 Ω /330 Ω divider or 1 k Ω pull up.

220 Ω /330 Ω (ISBC 901 OPTION)



Physical Characteristics

Width — 7.24 cm (2.85 in.)

Length — 9.40 cm (3.70 in.)

Height* — 2.04 cm (0.80 in.) ISBX 350 Board

— 2.86 cm (1.13 in.) ISBX 350 Board + Host Board

Weight — 51 gm (1.79 oz)

*See Figure 2.

Electrical Characteristics

DC Power Requirements

Power Requirement	Configuration
+ 5V @ 320 mA	Sockets XU3, XU4, XU5, and XU6 empty (as shipped).
+ 5V @ 500 mA	Sockets XU3, XU4, XU5, and XU6 contain 7438 buffers.
+ 5V @ 620 mA	Sockets XU3, XU4, XU5, and XU6 contain ISBC 901 termination devices.

Environmental

Operating Temperature — 0°C to 55°C

Reference Manual

9803191-01 — ISBX 350 Parallel I/O MULTIMODULE Manual (NOT SUPPLIED)

Reference Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Ave., Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBX 350	Parallel I/O MULTIMODULE Board



APPLICATION NOTE

AP-96

Interface Part No.	No. of Pins	Center Pin	Connector Type	Vendor	Part No.
Parallel IO Connector	3200	0.1	Female	IBM	3415-0001 with Earl
Parallel IO Connector	3200	0.1	Female	QTE	84001251A100

Note: Connector compatible with those listed may also be used.

Line Drivers and Terminators

IO Drivers — The following line drivers and terminators are all compatible with the IO driver sockets on the iSBX 350.

Part No.	Characteristics	Sink Current (mA)
7400	1	10
7401	1 OC	10
7402	1	10
7403	1	10
7404	1	10
7405	1 OC	10
7406	1	10
7407	1	10
7408	1	10
7409	1	10
7410	1	10
7411	1	10
7412	1	10
7413	1	10
7414	1	10
7415	1	10
7416	1	10
7417	1	10
7418	1	10
7419	1	10
7420	1	10
7421	1	10
7422	1	10
7423	1	10
7424	1	10
7425	1	10
7426	1	10
7427	1	10
7428	1	10
7429	1	10
7430	1	10
7431	1	10
7432	1	10
7433	1	10
7434	1	10
7435	1	10
7436	1	10
7437	1	10
7438	1	10
7439	1	10
7440	1	10
7441	1	10
7442	1	10
7443	1	10
7444	1	10
7445	1	10
7446	1	10
7447	1	10
7448	1	10
7449	1	10
7450	1	10
7451	1	10
7452	1	10
7453	1	10
7454	1	10
7455	1	10
7456	1	10
7457	1	10
7458	1	10
7459	1	10
7460	1	10
7461	1	10
7462	1	10
7463	1	10
7464	1	10
7465	1	10
7466	1	10
7467	1	10
7468	1	10
7469	1	10
7470	1	10
7471	1	10
7472	1	10
7473	1	10
7474	1	10
7475	1	10
7476	1	10
7477	1	10
7478	1	10
7479	1	10
7480	1	10
7481	1	10
7482	1	10
7483	1	10
7484	1	10
7485	1	10
7486	1	10
7487	1	10
7488	1	10
7489	1	10
7490	1	10
7491	1	10
7492	1	10
7493	1	10
7494	1	10
7495	1	10
7496	1	10
7497	1	10
7498	1	10
7499	1	10
7500	1	10

July 1980

Configuration	Power Requirement
Boards X04, X05, and X06 empty (as shipped)	+5V @ 320 mA
Boards X04, X05, and X06 contain 1455 buffers	+5V @ 500 mA
Boards X04, X05, and X06 contain iSBX 901 termination devices	+5V @ 520 mA

Environmental

Operating Temperature — 0°C to 55°C

Reference Manual

960391-01 — iSBX 350 Parallel IO MULTIMODULE Manual (NOT SUPPLIED)

Reference Manuals may be ordered from any Intel sales representative, distributor, or from Intel Literature Department, 3065 Bowers Ave., Santa Clara, California 95051.

Designing
iSBX™
Boards
MULTIMODULE™

Stephen Grubb
OEM Microcomputer
Systems Applications

INTRODUCTION

Intel's single board computers and the MULTIBUS™ system bus have become de facto industry standards in the microcomputer board market. The speed and capability of the bus coupled with the functionality and performance of the boards have been used to solve a large number of problems. iSBC products are in applications ranging from simple single board relay replacement to sophisticated multi-board business systems supporting large hard disk files. However, even with the range of functionality provided by standard iSBCs and expansion boards, designers have felt the need to design custom MULTIBUS-compatible boards to fit their application. Until the introduction of the iSBX concept, these custom boards had to be implemented using a separate MULTIBUS form factor board.

Intel has recently introduced a new line of board products and a new bus which are destined to become another industry standard because of the niche they fill. The new iSBX MULTIMODULE boards are designed to extend the functional capabilities of single board computers at a much lower cost than previously possible. iSBX MULTIMODULE boards are supported by a new bus — the iSBX bus, which allows the MULTIMODULE boards to be added directly to the on-board microprocessor bus. iSBX MULTIMODULE boards are from 10 to 20 square inches in size, therefore permitting small modular increments to a single board computer's capabilities.

System designers now have the capabilities of using either standard iSBCs or iSBX MULTIMODULE boards, or designing custom MULTIBUS compatible or iSBX MULTIMODULE boards. Cost-effective solutions are easily realized because of this added flexibility.

This application note discusses the iSBX MULTIMODULE concept, currently available MULTIMODULE boards and the iSBCs which support these boards. The iSBX bus interface specifications are discussed next, followed by consideration for designing custom iSBX MULTIMODULE boards. A specific design example using an Intel® 8279 Programmable Keyboard/Display Controller is presented.

The objective of the note is to introduce the reader to the iSBX MULTIMODULE concept for expanding iSBC functionality and to illustrate how a designer can effectively use this concept with either standard or custom iSBX boards.

References to further documentation on the iSBX bus, specific iSBX MULTIMODULE boards and iSBC host boards currently available may be found in the Related Intel Publications section in the front overleaf of this application note.

iSBX™ MULTIMODULE™ BOARD CONCEPT

The iSBX MULTIMODULE board concept was developed to provide the users of Intel single board computers (iSBCs) with a convenient method to incrementally expand the I/O or the computing capabilities of a single board computer. This expansion is done through the use of a new interface called the iSBX bus interface. This interface gives the user the capability of adding I/O mapped functions directly onto the microprocessor bus via plug-in modules that connect to the iSBC board by means of a special iSBX connector. With the use of this new bus interface, it is now possible to expand or add new features to your iSBC system without incurring large costs and long engineering development times.

There are a number of unique advantages to using the iSBX bus interface for system expansion rather than adding a separate expansion board to your system. First, when expansion is required, the user needs only to buy what is required for the application. Second, it is now possible to return to one board solutions for small systems. One board solutions eliminate the need for expensive backplanes and cardcages. Next, the iSBX interface connects directly to the microprocessor or local bus, as opposed to interfacing to the MULTIBUS system bus, therefore I/O expansion does not require system bus cycles. To the CPU, the iSBX board looks like any other on-board I/O device (Figure 1). Address decode logic exists on the iSBC host board for each iSBX connector on the host board.

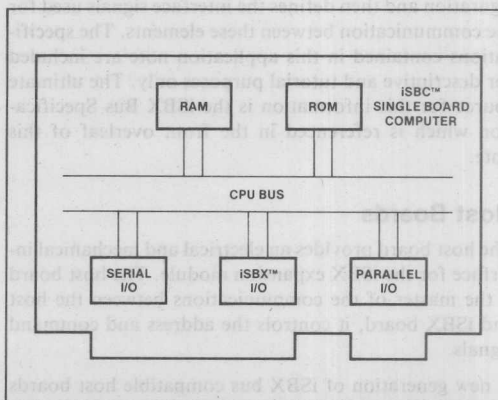


Figure 1. iSBC™ Host Board Block Diagram

Third, if there is no iSBC or MULTIBUS compatible expansion board available to fit the needs of your application or if the expansion boards available offer more capability than required, then it is possible to design a custom iSBX MULTIMODULE board. Custom iSBX boards offer several advantages over custom MULTIBUS boards: they require less board real estate (10 or 20

square inches versus 81 square inches) and less engineering design time; consequently, they cost considerably less to implement. Additional capability is therefore achieved with maximum productivity.

Currently available Intel iSBX MULTIMODULE Boards include:

- 1) iSBX 350 Parallel I/O MULTIMODULE board which contains 24 programmable I/O lines with sockets for line drivers and terminators.
- 2) iSBX 351 Serial I/O MULTIMODULE board containing one RS232 or RS449/422 programmable synchronous/asynchronous communications channel and two timers.
- 3) iSBX 331 Fixed/Floating Point Math MULTIMODULE board which permits fixed or floating point mathematics via the Intel 8231 device.
- 4) iSBX 332 Floating Point Math MULTIMODULE board which permits floating point mathematics using the Intel and proposed IEEE floating point standards via an Intel 8232 device.

With these iSBX MULTIMODULE boards and other soon-to-be-announced boards, the capability now exists to economically tailor a single board computer to the application using off-the-shelf products.

iSBX™ MULTIMODULE™ SYSTEM INTERFACE

This section begins by describing the basic system elements used in an iSBX MULTIMODULE interface configuration and then defines the interface signals used for the communication between these elements. The specifications contained in this application note are included for descriptive and tutorial purposes only. The ultimate source for this information is the iSBX Bus Specification which is referenced in the front overleaf of this note.

Host Boards

The host board provides an electrical and mechanical interface for the iSBX expansion module. The host board is the master of the communications between the host and iSBX board, it controls the address and command signals.

A new generation of iSBX bus compatible host boards are evolving. The first board available from Intel is the iSBC 80/10B Single Board Computer. The 80/10B contains an 8080A CPU operating at 2 MHz, 1K bytes of RAM with sockets available for expansion to 4K bytes of RAM, sockets for up to 16K bytes of EPROM, 24 parallel I/O lines, a programmable synchronous/asynchronous communications interface and a fixed 1.04 msec timer. The 80/10B has one iSBX connector, permitting the use of an iSBX MULTIMODULE board.

The second iSBC board available supporting iSBX boards is the iSBC 80/24 Single Board Computer. The 80/24 board, which supports two iSBX MULTIMODULE boards, contains an 8085A-2 CPU operating at 4.8 or 2.4 MHz, 4K bytes of RAM, sockets for up to 32K bytes of EPROM, 48 parallel I/O lines, a programmable synchronous/asynchronous communications interface, three programmable interval timers and a programmable interrupt controller. Further RAM expansion on the 80/24 board is accomplished by the addition of an iSBC 301 4K byte RAM MULTIMODULE board which expands the RAM by an additional 4K bytes for a total of 8K bytes. The iSBC 301 MULTIMODULE board is not iSBX bus compatible; it is attached via pins and sockets in the RAM section of the host board.

iSBX™ MULTIMODULE™ Boards

The iSBX MULTIMODULE boards communicate with the host boards via the iSBX bus interface. These iSBX boards are I/O mapped through pre-defined select lines to specific port addresses. The iSBX bus currently defines an 8-bit data path compatible with both 8 and 16-bit future iSBC host boards. Examples of possible iSBX expansion boards include a floppy disk controller, a cassette interface, analog-to-digital converter or digital-to-analog converter boards, an interface to the IEEE 488 Bus and a video graphics display interface board.

There are two standard sizes of iSBX boards: a single-wide board measuring 7.24 by 9.40 cm (2.85 by 3.70 inches) and a double-wide board measuring 7.24 by 19.05 cm (2.85 by 7.50 inches). The iSBX MULTIMODULE boards mount onto any microcomputer board containing an iSBX connector and mounting hole. The iSBX boards physically plug into the iSBX connector on the host board and are secured with a nylon stand-off and screws. The mounting hardware supplied as part of the iSBX board includes:

- 1) One nylon spacer, 1/2" threaded
- 2) Two nylon screws, 1/4" 6-32
- 3) One 36-pin connector, factory-installed onto the iSBX module. (These may also be purchased from Intel.)

The interconnection between the host board and iSBX board, as well as the mounting clearances, may be seen in Figures 2 and 3.

NOTE

The iSBX board, when installed onto a host board, occupies an additional card slot adjacent to the base board in an iSBC 604/614 Cardcage. However, the base board may be inserted in the top card slot of the cardcage. If this is done, no additional slots are required.

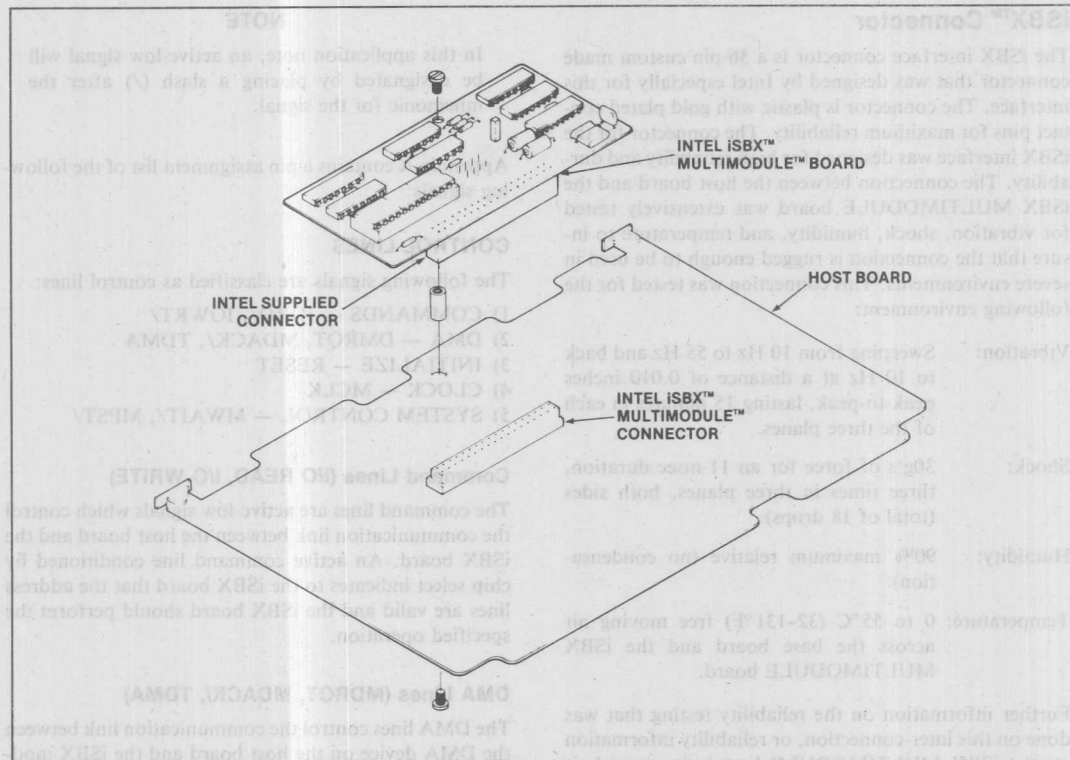


Figure 2. Connection of iSBX™ MULTIMODULE™ to Host Board

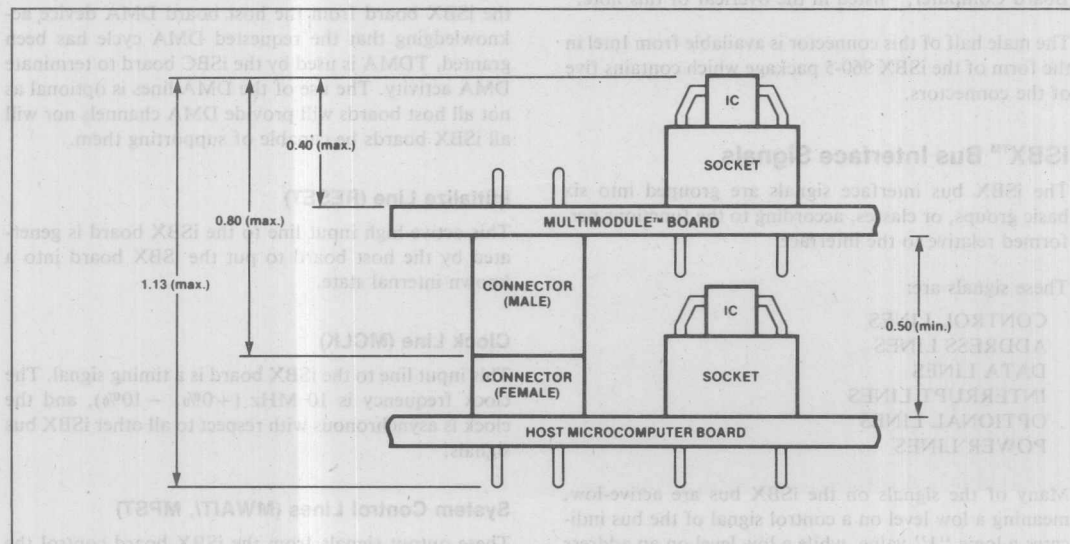


Figure 3. iSBX™/iSBC™ Mounting Clearance (inches)

ISBX™ Connector

The iSBX interface connector is a 36-pin custom made connector that was designed by Intel especially for this interface. The connector is plastic with gold plated contact pins for maximum reliability. The connector for the iSBX interface was designed for high reliability and durability. The connection between the host board and the iSBX MULTIMODULE board was extensively tested for vibration, shock, humidity, and temperature to insure that the connection is rugged enough to be used in severe environments. This connection was tested for the following environment:

Vibration:	Sweeping from 10 Hz to 55 Hz and back to 10. Hz at a distance of 0.010 inches peak-to-peak, lasting 15 minutes in each of the three planes.
Shock:	30g's of force for an 11-msec duration, three times in three planes, both sides (total of 18 drops).
Humidity:	90% maximum relative (no condensation).
Temperature:	0 to 55°C (32–131°F) free moving air across the base board and the iSBX MULTIMODULE board.

Further information on the reliability testing that was done on this inter-connection, or reliability information on the iSBX MULTIMODULE boards in general, is contained in the Reliability Report, RR-29, "Intel iSBX MULTIMODULE Boards and iSBC 80/10B Single Board Computer," listed in the overleaf of this note.

The male half of this connector is available from Intel in the form of the iSBX 960-5 package which contains five of the connectors.

iSBX™ Bus Interface Signals

The iSBX bus interface signals are grouped into six basic groups, or classes, according to the functions performed relative to the interface:

These signals are:

CONTROL LINES
ADDRESS LINES
DATA LINES
INTERRUPT LINES
OPTIONAL LINES
POWER LINES

Many of the signals on the iSBX bus are active-low, meaning a low level on a control signal of the bus indicates a logic "1" value, while a low level on an address or data signal of the bus represents a logic "0" value.

NOTE

In this application note, an active-low signal will be designated by placing a slash (/) after the mnemonic for the signal.

Appendix A contains a pin assignment list of the following signals:

CONTROL LINES

The following signals are classified as control lines:

- 1) COMMANDS — IORD/, IOWRT/
- 2) DMA — DMRQT, MDACK/, TDMA
- 3) INITIALIZE — RESET
- 4) CLOCK — MCLK
- 5) SYSTEM CONTROL — MWAIT/, MPST/

Command Lines (I/O READ, I/O WRITE)

The command lines are active-low signals which control the communication link between the host board and the iSBX board. An active command line conditioned by chip select indicates to the iSBX board that the address lines are valid and the iSBX board should perform the specified operation.

DMA Lines (MDRQT, MDACK/, TDMA)

The DMA lines control the communication link between the DMA device on the host board and the iSBX module. DMRQT is an active-high output signal from the iSBX board to the host board's DMA device requesting a DMA cycle. MDACK/ is an active-low input signal to the iSBX board from the host board DMA device acknowledging that the requested DMA cycle has been granted. TDMA is used by the iSBC board to terminate DMA activity. The use of the DMA lines is optional as not all host boards will provide DMA channels nor will all iSBX boards be capable of supporting them.

Initialize Line (RESET)

This active-high input line to the iSBX board is generated by the host board to put the iSBX board into a known internal state.

Clock Line (MCLK)

This input line to the iSBX board is a timing signal. The clock frequency is 10 MHz (+0%, -10%), and the clock is asynchronous with respect to all other iSBX bus signals.

System Control Lines (MWAIT/, MPST)

These output signals from the iSBX board control the state of the system. Active MWAIT/ (active-low) will

put the CPU on the host board into a wait state, providing additional time for the iSBX board to perform the requested operation. MPST/ is an active-low signal (usually tied to signal ground) that informs the host board I/O decode logic that an iSBX module has been installed.

ADDRESS AND CHIP SELECT LINES

The address and chip select lines are made up of the following signals:

- 1) ADDRESS LINES — MA0, MA1, MA2
- 2) CHIP SELECT LINES — MCS0/, MCS1/

Address Lines (MA0, MA1, MA2)

These active-high input lines to the iSBX boards are generally the least three significant bits of the I/O addresses. In conjunction with the command and chip select lines, they establish the I/O port address being accessed.

Chip Select Lines (MCS0/, MCS1/)

These active-low input lines to the iSBX board are the result of the host board I/O decode logic. When active, the MCS/ lines condition the I/O command signals and thus enable communication between the iSBX board and the host board.

DATA LINES (MD0-MD7)

There are eight bidirectional data lines. These active-high lines are used to transmit or receive information to or from the iSBX ports. MD0 is the least significant bit.

INTERRUPT LINES (MINTR0, MINTR1)

These active-high output lines from the iSBX board are used to make interrupt requests to the host board. These lines are jumper enabled and disabled on the host board via wire wrap posts.

OPTION LINES (OPT0, OPT1)

These two signals are reserved lines that are connected to wire wrap posts on both the host board and the iSBX MULTIMODULE board. They are for unique requirements where a user needs a host board or MULTIBUS bus signal on the iSBX module.

POWER LINES

All host boards provide +5 volts as well as ± 12 volts to the iSBX MULTIMODULE board along with signal ground. All power supply voltages are $\pm 5\%$. Table 1 gives the power supply specifications for the iSBX interface.

Table 1. Power Supply Specifications

Minimum (volts)	Nominal (volts)	Maximum (volts)	Maximum (current)*
+4.75	+5.0	+5.25	3.0A
+11.4	+12	+12.6	1.0A
-12.6	-12	-11.4	1.0A
—	GND	—	6.0A

*Per iSBX MULTIMODULE board mounted on base board.

iSBX™ BUS INTERFACING

This section of the application note focuses on the iSBX interface and design considerations related to interfacing with the iSBX bus. It discusses the way the major operations like READ, WRITE, and DMA work, and the timing diagrams associated with each. There is also a discussion on other considerations for designing with the iSBX bus.

Bus Timing

The AC timing specifications for the iSBX bus interface can be found in Appendix B of this application note. It should be emphasized that the interface timing between the host board and the iSBX MULTIMODULE board is very critical. This is largely due to the fact that the iSBX board is attached directly to the microprocessor bus. If the timing specifications are not met, unpredictable and possibly intermittent operation of the host board may result.

Command Operations

The command lines (IORD/, IOWRT) are driven from the host board by three-state drivers with pull-up resistors or standard TTL totem-pole drivers. These lines indicate to the iSBX board that action is being requested. There are two types of operations for each command line and it is the iSBX board that determines which operation is to be performed.

READ OPERATIONS (IORD/)

Two different types of read operations are possible. The first type of read is called a full speed I/O READ. The host board generates a valid I/O address (MA0-MA2) and a valid chip select signal (MCS1/) which is then sent to the iSBX board; after the set-up times are met, the host board activates the IORD/ line. At this time, the iSBX board must generate valid data from the addressed I/O port in less than 250 ns. The host board then reads the data and removes the READ command, address and chip selects. These are shown in the timing diagram for this operation (Figure 4). The second type of read operation is called an I/O READ with Wait. This READ is used by iSBX boards that cannot perform a full speed read operation. Under this operation the

host board generates the valid address and chip select signals, as in the full speed read. But this time the iSBX board will activate the MWAIT/ signal, which in turn removes the READY input to the CPU, putting it into a Wait state. The CPU, however, first activates the IORD/ signal before going into the Wait state. After valid data is placed on the iSBX data bus by the iSBX board, the iSBX board will remove the MWAIT/ signal. The host board will then read the data and remove the command, address, and chip select lines. This I/O READ with Wait operation is shown in Figure 5.

WRITE OPERATIONS (IOWRT/)

There are also two types of write operations possible: the type performed is again determined by the iSBX board. In the full speed I/O WRITE operation, the host board generates a valid I/O address and chip select and then activates the IOWRT/ line after the necessary set-up times are met. The IOWRT/ line, after being activated, will remain active for 300 ns and the data will be valid for 250 ns before the IOWRT/ command is re-

moved. The host board will then remove the data, address, and chip select lines after the hold times are met, as shown in the timing diagram of this operation (Figure 6).

This second write operation is the I/O WRITE with Wait operation. This WRITE is used by the iSBX boards that cannot write into an I/O port with the full speed write specifications. The host board again generates valid address and chip select signals as in the full speed write operation. However, this time the iSBX board generates the MWAIT/ signal based on address information (chip select and MA0-MA1). The activation of MWAIT/ causes the removal of READY to the CPU, thus causing the CPU to go into a Wait state. The iSBX board removes the MWAIT/ signal (allowing the CPU to leave its Wait state) when it has satisfied the WRITE pulse width requirements. At this time the board removes the WRITE command, followed by the data, address, and chip select lines. This I/O WRITE with Wait operation can be seen in Figure 7.

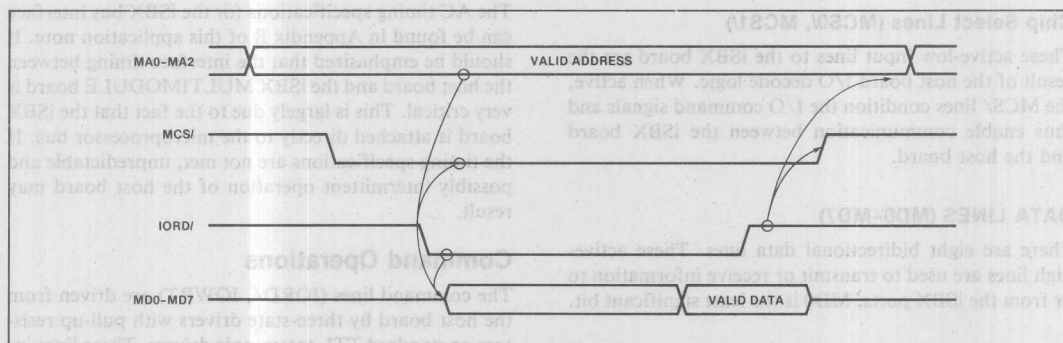


Figure 4. Full Speed I/O Read Operation

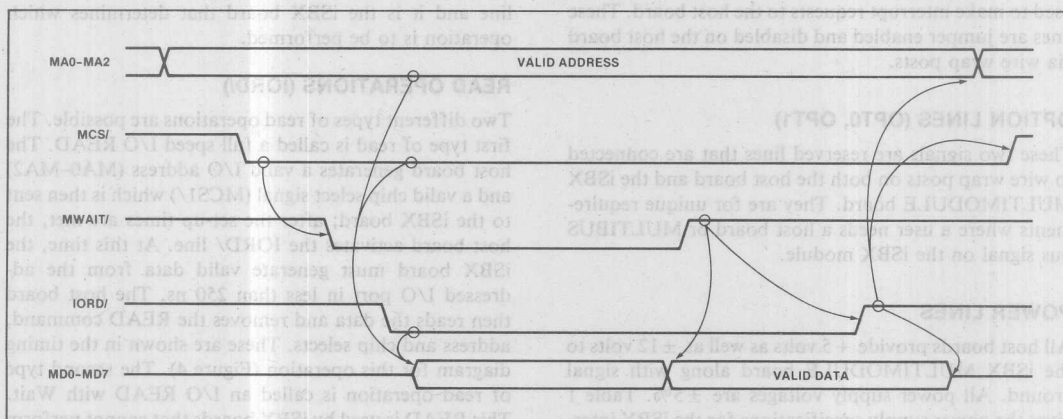


Figure 5. I/O Read with Wait Operation

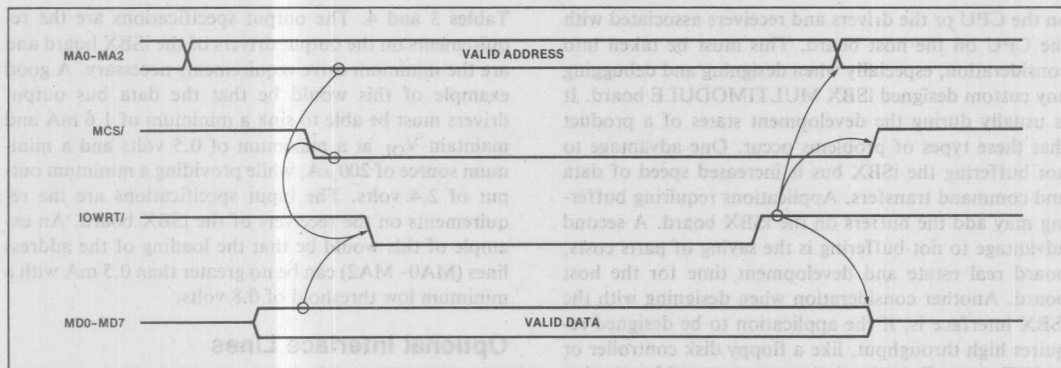


Figure 6. Full Speed I/O Write Operation

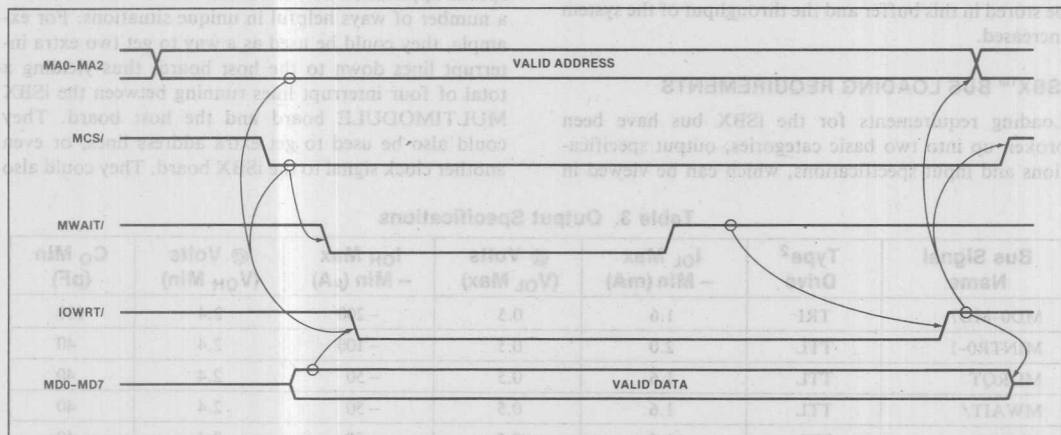


Figure 7. I/O Write with Wait Operation

iSBX™ Addressing

The iSBX boards are addressed by the host board through the use of the address lines MA0, MA1 and MA2, and the chip select lines MCS0/ and MCS1/. The host board decodes the I/O addresses and in turn generates the chip selects for the iSBX boards. In an 8-bit system the host board decodes the high order 13 address bits and generates the appropriate chip select corresponding to those address bits. The low order three address bits are passed to the iSBX board via MA0-MA2. Thus, a host board reserves two blocks of eight I/O ports for each iSBX connector. There can be as many as three iSBX connectors per host board, therefore a total of 48 addresses or six blocks of eight I/O ports that can be reserved for the iSBX boards. Table 2 contains a list of the I/O addresses and their corresponding host board iSBX port assignments of the iSBC 80/10B and iSBC 80/24 host boards.

Table 2. iSBX™ Host Board Port Assignment

iSBX™ Connector Number	Chip Select	iSBX™ Port Addresses
iSBC 80/10B Connector	MCS0/ MCS1/	F0-F7 F8-FF
iSBC 80/24 First Connector	MCS0/ MCS1/	F0-F7 F8-FF
iSBC 80/24 Second Connector	MCS0/ MCS1/	C0-C7 C8-CF

Considerations for iSBX™ Bus Interfacing

When designing with the iSBX interface it is important to note that the iSBX bus is not buffered on the host board. Since there is no isolation between the iSBX board and the host board CPU bus, a short between signal lines and power or ground could have a direct effect

on the CPU or the drivers and receivers associated with the CPU on the host board. This must be taken into consideration, especially when designing and debugging any custom designed iSBX MULTIMODULE board. It is usually during the development states of a product that these types of problems occur. One advantage to not buffering the iSBX bus is increased speed of data and command transfers. Applications requiring buffering may add the buffers on the iSBX board. A second advantage to not buffering is the saving of parts costs, board real estate and development time for the host board. Another consideration when designing with the iSBX interface is, if the application to be designed requires high throughput, like a floppy disk controller or a CRT controller, the designer may consider putting some type intelligent control of buffer RAM onto the iSBX board. By doing this, the transfer information can be stored in this buffer and the throughput of the system increased.

iSBX™ BUS LOADING REQUIREMENTS

Loading requirements for the iSBX bus have been broken up into two basic categories, output specifications and input specifications, which can be viewed in

Tables 3 and 4. The output specifications are the requirements on the output drivers of the iSBX board and are the minimum drive requirements necessary. A good example of this would be that the data bus output drivers must be able to sink a minimum of 1.6 mA and maintain V_{OL} at a maximum of 0.5 volts and a minimum source of $200 \mu A$, while providing a minimum output of 2.4 volts. The input specifications are the requirements on the receivers of the iSBX board. An example of this would be that the loading of the address lines (MA0- MA2) can be no greater than 0.5 mA with a minimum low threshold of 0.8 volts.

Optional Interface Lines

The iSBX interface has two optional lines which were included for the user to configure the iSBX board for special application needs. These two lines can be used in a number of ways helpful in unique situations. For example, they could be used as a way to get two extra interrupt lines down to the host board, thus yielding a total of four interrupt lines running between the iSBX MULTIMODULE board and the host board. They could also be used to get extra address lines, or even another clock signal to the iSBX board. They could also

Table 3. Output Specifications

Bus Signal Name	Type ² Drive	I_{OL} Max - Min (mA)	@ Volts (V_{OL} Max)	I_{OH} Max - Min (μA)	@ Volts (V_{OH} Min)	C_O Min (pF)
MD0-MD7	TRI	1.6	0.5	- 200	2.4	130
MINTR0-1	TTL	2.0	0.5	- 100	2.4	40
MDRQT	TTL	1.6	0.5	- 50	2.4	40
MWAIT/	TTL	1.6	0.5	- 50	2.4	40
OPT1-2	TTL	1.6	0.5	- 50	2.4	40
MPST/	TTL	Note 3				

Table 4. Input Specifications

Bus Signal Name	Type ² Receiver	I_{IL} Max (mA)	@ Volts (V_{IN} Max)	I_{IH} Max (μA)	@ Volts (V_{IN} Min)	C_I Max (pF)
MD0-MD7	TRI	- 0.5	0.4	70	2.4	40
MA0-MA2	TTL	- 0.5	0.4	70	2.4	40
MCS0/-MCS1/	TTL	- 4.0	0.4	100	2.4	40
MRESET	TTL	- 2.1	0.4	100	2.4	40
MDACK/	TTL	- 1.0	0.4	100	2.4	40
IORD/ IOWRT/	TTL	- 1.0	0.4	100	2.4	40
MCLK	TTL	2.4	0.4	100	2.4	40
OPT1-OPT2	TTL	2.0	0.4	100	2.4	40

NOTES:

1. Per iSBX MULTIMODULE board.
2. TTL = standard totem-pole output. TRI = three-state.
3. iSBX MULTIMODULE board must connect this signal to ground.

be used to send a special status line to or from the iSBX MULTIMODULE board.

iSBX™ MULTIMODULE™ DESIGN EXAMPLE

This section covers the description of a custom iSBX MULTIMODULE board which uses the Intel 8279 Programmable Keyboard/Display Controller. This iSBX board, when added to an iSBC host board, provides an interface to a keyboard and display. A description of the hardware design considerations for breadboarding the hardware is presented. Following this, a software exerciser, useful for debugging the board, is described. A listing for the exerciser is contained in Appendix C.

Since the iSBX MULTIMODULE board was designed using the Intel 8279 Programmable Keyboard/Display Controller, a brief description of the 8279 is presented. The 8279 is a general purpose programmable keyboard and display I/O controller which was designed for use with the Intel microprocessors. The keyboard portion of this device is capable of providing a scanned interface to a 64-contact key matrix. It is also possible to interface to an array of sensors or a strobed keyboard, such as those of the Hall Effect or the ferrite variety. The 8279 provides a variety of keyboard inputs (i.e., 2-key lockout and N-key rollover), and all key entries are debounced

and strobed into an 8-character FIFO. The display portion provides the user with a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used, as well as simple indicators. The 8279 is used in this iSBX design example to provide an interface of 2-key lockout with key debounce to a 64-character keyboard, and an interface for a 16-character, 18-segment alphanumeric display.

iSBX™ MULTIMODULE™ Board Design

The iSBX board that was designed for this application note contains a total of three IC's, the keyboard/display controller, a flip-flop, and a 3-to-8-line decoder. Figure 8 contains a block diagram of the hardware used in this design example. Figure 9 contains a schematic for the portion of the design example resident on the custom iSBX board.

The design offers the user some flexibility as to the type of display or keyboard to be attached. For example, if the application design was defined to be for a 7-segment, 16-character display (as the 8279 is designed to drive), a 4-to-16-line decoder along with the display drivers could be added to the iSBX board. Another idea would be to include everything except the display drivers and the display on the iSBX board, and to put the dis-

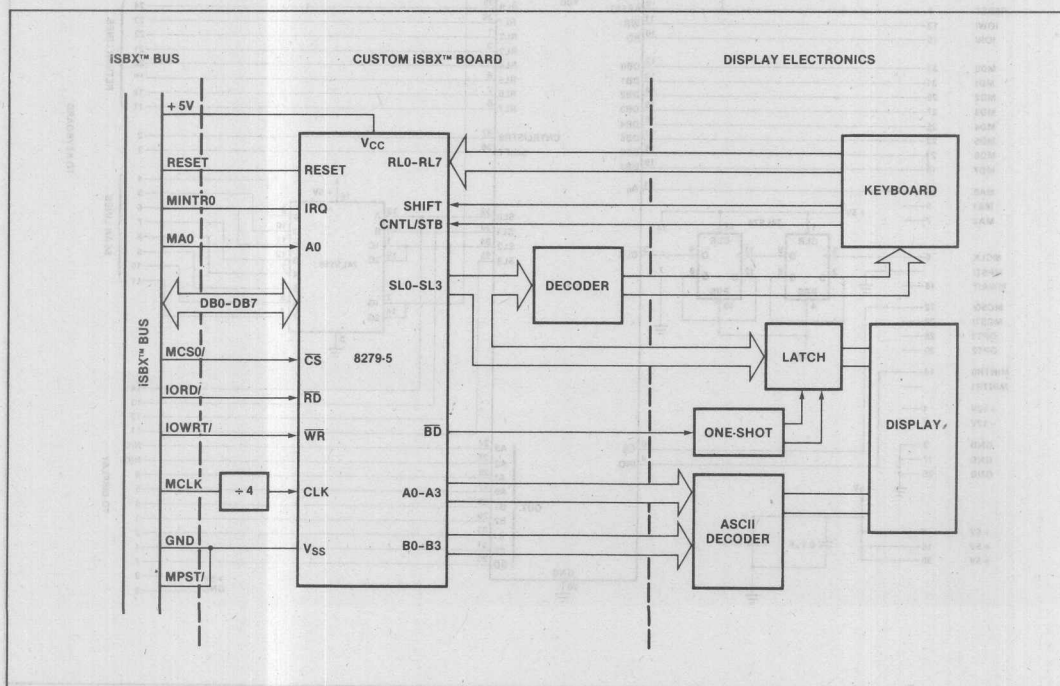


Figure 8. Block Diagram of the iSBX™ Design Example

play and drivers in with the keyboard. It is possible, and probably desirable in some applications, to incorporate some of the display electronics onto the iSBX MULTI-MODULE board. Some of the IC's found in the display portion of this design could also have been placed on the iSBX board, as there is enough room on the finished product for doing so.

The design was very easy to implement because, with the exception of one signal, all of the iSBX bus signals necessary to drive the 8279 are connected directly without any extra logic needed. The one signal that would not connect directly to the interface is the clock signal MCLK from the bus to CLK on the controller. It is not possible to connect these two together as MCLK is a 10 MHz signal and the 8279 requires a maximum clock signal of 3.1 MHz to generate its internal timings. It is necessary to add a 74LS74 dual D-type flip-flop to divide the MCLK signal by 4 for the controller. With this exception, all other signals, DB0-DB7 to MD0-MD7, A₀ to MA0, CS/ to MCS0/, etc., are connected directly

to the iSBX interface. To meet the timing requirements of the iSBX bus, a high speed version of the 8279, the 8279-5, is used.

The keyboard interface side of the iSBX board consists of a 3-to-8-line decoder, which is used for scanning the keyboard matrix. The 8279 scan lines SL0-SL2 are decoded by a 74LS156 open-collector output decoder and sent to the keyboard via a connector.

The display interface of the iSBX board consists of sending the scan lines and the display outputs to the display module via a connector. The scan lines SL0-SL3 are sent to the display drivers, and the display outputs A0-A3 and B0-B3 are sent to an ASCII to 18-segment decoder driver. The display is discussed in further detail in the next section of this application note.

Display Module Design

The display module design (Figure 10) consists of two 8-digit HDSP 6805 Alphanumeric Displays by Hewlett

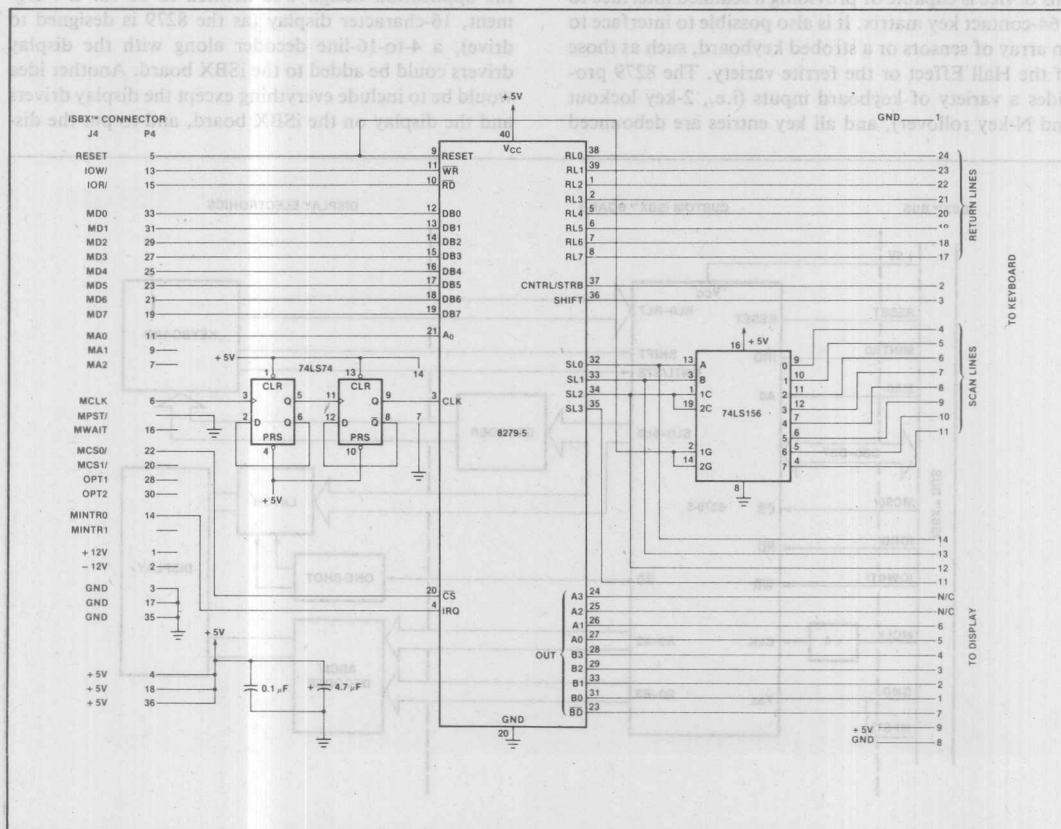
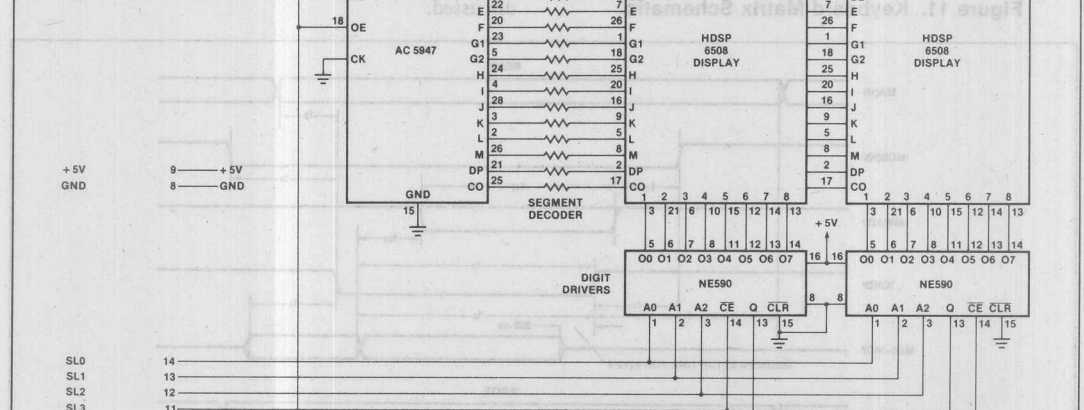


Figure 9. Schematic of the custom iSBX™ Board

Pin connection diagram for the LS1C 80108 board. The diagram shows a 16-pin connector labeled "DISPLAY CONNECTOR (FROM ISBX* BOARD)" on the left and a 16-pin connector labeled "Vcc" on the right. The pins are numbered 1 through 16. The connections are: Pin 1 to A0, Pin 2 to B1, Pin 3 to B2, Pin 4 to A0, Pin 5 to A3, Pin 6 to A1, Pin 7 to A5, Pin 8 to A15, Pin 9 to A19, Pin 10 to A0, Pin 11 to A1, Pin 12 to A2, Pin 13 to A3, Pin 14 to A13, Pin 15 to A15, Pin 16 to A19. A 3.9K resistor is connected between pins 7 and 8. A +5V supply is connected to pin 16.



ample. A diagram of the keyboard interfaces and matrix can be seen in Figure 11.

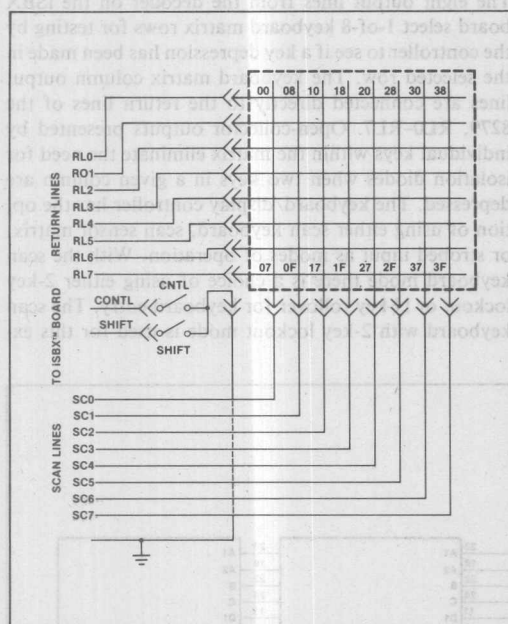


Figure 11. Keyboard Matrix Schematic

Operation with the iSBC 80/10B™ Single Board Computer

The 8279 on the iSBX expansion board is initialized to its mode of operation following a system reset. The keyboard mode of operation is to scan the keyboard with 2-key lockout, and the display mode is set for the 16-character left entry mode of operation. Upon receiving a character from the keyboard, the 8279 generates an interrupt along the MINTR0 line of the iSBX bus to the CPU. At this time the iSBC 80/10B board commences I/O read operations to the iSBX board by generating valid I/O address and chip select commands on the MA0 and MCS0/ signal lines. After the setup times are met, the 80/10B issues an I/O read command by asserting the IORD/ line on the bus, and the base board reads the data from the iSBX board and removes the IORD/, MA0, and MCS0/ signals from the bus. After the data has been read in from the keyboard, it must be output to the display. The iSBC 80/10B board starts an I/O write operation by generating a valid I/O address and the chip select signal with the MA0 and MCS0/ lines. After the valid setup times are met, the IOWRT/ line is activated by the base board. When the data has been valid for a minimum of 250 ns, the host board removes the IOWRT/ line. When the hold times have been met, the data, address and chip select lines are also removed. Figure 12 shows the timing diagrams just discussed.

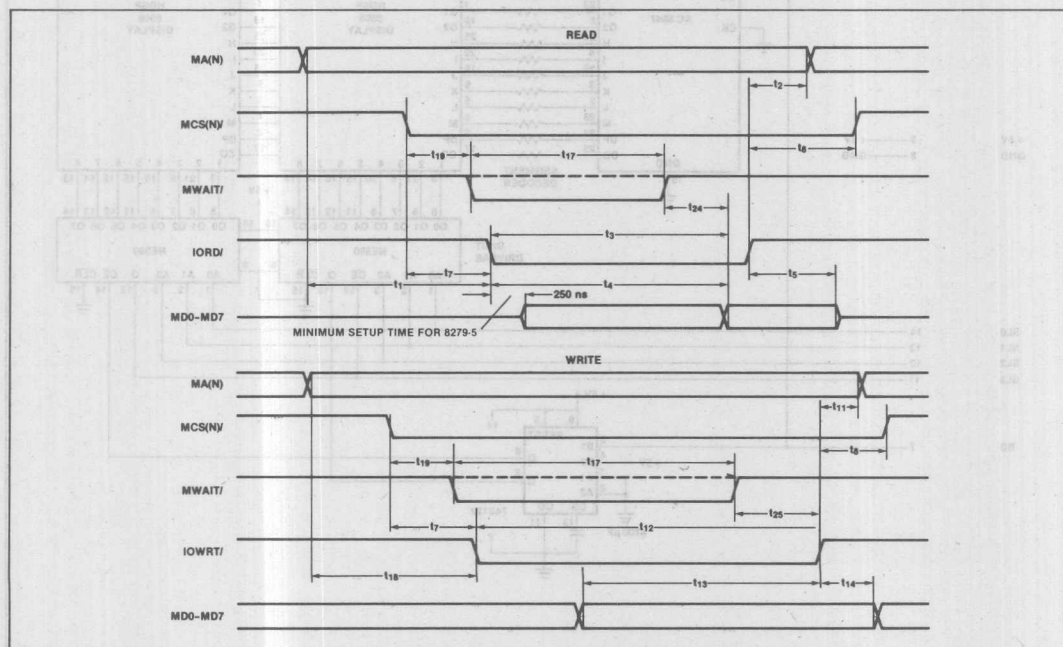


Figure 12. System Timing Diagrams

Breadboarding the Design

When doing the layout of the breadboard, it is also necessary to take into consideration the space required by the mounting holes and to plan the positioning of the components accordingly. (This information is available in the iSBX Bus Specification Manual.)

When attaching the breadboarded design, which typically contains raised wirewrap posts, it is necessary to raise the breadboard well above the host board. This can be accomplished by building a small cable and putting the breadboard on longer nylon standoffs. It is not recommended that the cable be longer than 15 cm (6 in.), otherwise bus timing problems could result.

With the breadboarding finished it is a good idea to re-check all wiring connections for possible errors. Also check all signal lines with an ohmmeter between power, and then ground, for potential shorts. An error at this point can cause serious damage to the host board!

Software Considerations

The software written for this application is an exerciser that is used for hardware checkout. It is a small program designed to echo characters from the keyboard to the display. The software was edited, assembled, linked and located with an Intel development system; it was then debugged with an in-circuit emulator. Both the software and the hardware debug is covered in the next section of this application note.

To facilitate this discussion the software exerciser is divided into three sections based upon the functions performed. The three functions are:

- 1) Keyboard interrupt routine
- 2) Initialization and flag checking routine
- 3) Character output routine

A complete listing of the software exerciser can be found in Appendix C.

KEYBOARD INTERRUPT ROUTINE

The 8279 generates an interrupt to the CPU whenever data is introduced into its FIFO/Sensor RAM. The interrupt is cleared by doing a data read. Whenever a key on the keyboard is depressed an interrupt is generated. Two things are required when an interrupt occurs. First, the keyboard input data must be retrieved and stored. Second, the interrupt routine must indicate that there is some data ready to be output to the display. Therefore, a buffer is created in memory (called "BUFF") at location 3C00H to store the keyboard data. A data present flag is set in a register (REG. C) to indicate that data is ready to be output and can be found in the buffer. In this way the interrupt routine is used to input characters from the keyboard to the input buffer. The buffer is then read by the output routine, which sends the characters to the display.

INITIALIZATION AND FLAG CHECKING ROUTINE

The initialization and flag checking routine first sets the stack pointer to the top of memory. After this the program proceeds to initialize the 8279 Keyboard/Display Controller to its proper mode of operation. The modes of operation used for this application note is scanned keyboard with 2-key lockout for the keyboard, and 16 characters with left entry for the display. As the 8279 has a desired internal operating frequency of 100 kHz, the frequency divider chain is programmed to divide by 19 hex, or 25 decimal. After the 8279 has been initialized, the program begins its next procedure of clearing the buffers. The keyboard input buffer, "BUFF", as well as the display buffer, "DBUFF", are both cleared to a blank display. This is done so that at the time of power up, the display will come up blank. With the initialization now complete, the program disables the interrupts and checks the data present flag for an indication that data might be present for output. If the data present flag is set, the output character routine is called; if it is not set, the interrupts are enabled and the program loops back around to check again. In summary, this routine initializes the 8279 and clears the buffers, and then loops on the data present flag looking for an indication that data is present in the input buffer. The input buffer is a one-byte wide buffer named "BUFF."

CHARACTER OUTPUT ROUTINE

The character output routine brings the character in from "BUFF" (the keyboard input buffer) and compares it to the characters located in a table. If the character can be matched to a character in the table it is replaced in "BUFF" with the corresponding character located in the same position of a second table. If there is no match, it is compared to the code for a control character. If there is no match with a control character, a compare is made to see if the character is a delete character. When a match is found and the acceptable character is placed in "BUFF", the output routine shifts the data in the display buffer (Figure 13) one position to the left and places the character from the input buffer into the display buffer at position "DBUFF" + 15. Now that

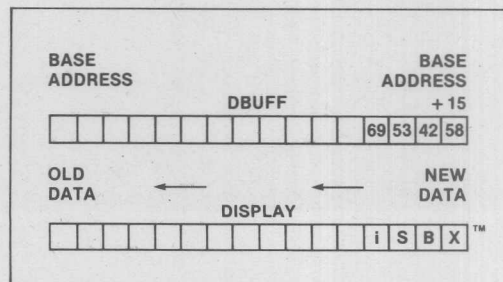


Figure 13. Display Buffer

the new information is in the display, the routine copies the complete contents of the display buffer, "DBUFF", to "DBUFF" + 15 to the display. In the case of the input character being matched up with a delete character, all information in the display buffer is shifted to the right one position and the ASCII code for a blank character is placed into the left-most position or the base address of "DBUFF", thus making the next character sent to the display a blank character. In the case of a control character, nothing is done and the program returns to the flag checking routine.

Debug Considerations

Hardware and software debug was accomplished using an iSBC 80/10B Single Board Computer, an iSBC 655 Chassis, an Intellec® Series II Model 230 Microcomputer Development System, and an ICE-80™ In-Circuit Emulator.

The software was down-loaded from the disk to the iSBC 80/10B board using the in-circuit emulator. The ICE™ module gives the engineer the capability of interrogating the iSBC system by allowing the user to access and display the CPU register contents, status, system memory contents, and all I/O devices and their data.

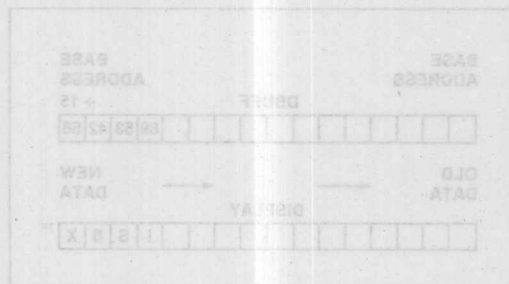
The iSBC 80/10B board was configured to enable interrupts from the iSBX board via the interrupt 0 line (MINTR0), which is connected to the interrupt pin of the 8080 CPU. The iSBX board was attached to the iSBC 80/10B board via the iSBX connector. The iSBC 80/10B board was powered-up and the iSBX board was

checked for proper power and ground connections. The ICE-80 emulator was connected to the iSBC 80/10B board. Using the interrogation mode of the emulator, it is possible to check proper functioning of the iSBX board by sending and receiving data to/from the 8279. The keyboard can be tested by depressing a key on the keyboard and then examining the FIFO/Sensor RAM to see if the data was entered. The display RAM can also be read and written to for testing the interface to the display.

After this initial checking of the iSBX board, the software exerciser can then be down-loaded with the ICE module to further check the board.

SUMMARY

The objective of this application note is to introduce the reader to the iSBX MULTIMODULE concept for expanding a single board computer's functionality, and to illustrate how a designer can use this concept with either standard or custom iSBX boards. In contrast to system expansion using MULTIBUS-compatible boards, iSBX MULTIMODULE boards provide smaller, lower cost, incremental expansion. This application note explains how a custom iSBX board can be designed and debugged. Using this capability, it is now possible to more quickly add new VLSI technology to systems as the technology becomes available. Intel will continue to provide new iSBX MULTIMODULE boards and, because of the the publication of the iSBX Bus Specification and this application note, it will be easier for Intel's customers to also design and build their own custom iSBX boards.



KEYBOARD INTERRUPT ROUTINE

The 8279 generates an interrupt to the CPU whenever data is introduced into its FIFO/Sensor RAM. The interrupt is cleared by doing a data read. Whenever a key is depressed on the keyboard, an interrupt is generated. Two things are required when an interrupt occurs. First, the keyboard input data must be retrieved and stored. Second, the interrupt routine must indicate that there is some data ready to be output to the display. Therefore, a buffer is created in memory (called "DBUFF") at location 100H to store the keyboard data. A data present flag is set in a register (REG C) to indicate that data is ready to be output and can be found in the buffer. In this way, the interrupt routine is used to input characters from the keyboard to the input buffer. The buffer is then read by the output routine, which sends the char-

APPENDIX A ISBX™ SIGNAL PIN ASSIGNMENTS

Pin	Mnemonic	Description	Pin	Mnemonic	Description
38	GND	Signal Ground	38	+3V	+3 Volts
37	MD0	MDATA Bit 0	34	MDROT	M DMA Request
36	MD1	MDATA Bit 1	33	MDACK	M DMA Acknowledge
35	MD2	MDATA Bit 2	30	OPT0	Option 0
34	MD3	MDATA Bit 3	28	OPT1	Option 1
33	MD4	MDATA Bit 4	26	TDMA	Terminate DMA
32	MD5	MDATA Bit 5	24		Reserved
31	MD6	MDATA Bit 6	23	MC20	M Chip Select 0
30	MD7	MDATA Bit 7	20		Reserved
29	GND	Signal Ground	19		Reserved
28	RD	Read Command	18		Reserved
27	RDWR	Read/Write Command	17		Reserved
26	MA0	M Address 0	16		Reserved
25	MA1	M Address 1	15		Reserved
24	MA2	M Address 2	14	MRSTV	ISBX MULTIMODULE Board Present
23	RESET	Reset	13	MCLE	M Clock
22	GND	Signal Ground	12		+3 Volts
21	+12V	+12 Volts	11		-12 Volts

Pin assignments are reserved for future use.

APPENDIX A	1-192
APPENDIX B	1-193
APPENDIX C	1-194

APPENDIX A

iSBX™ SIGNAL PIN ASSIGNMENTS

Pin	Mnemonic	Description	Pin	Mnemonic	Description
35	GND	Signal Ground	36	+ 5V	+ 5 Volts
33	MD0	MDATA Bit 0	34	MDRQT	M DMA Request
31	MD1	MDATA Bit 1	32	MDACK/	M DMA Acknowledge
29	MD2	MDATA Bit 2	30	OPT0	Option 0
27	MD3	MDATA Bit 3	28	OPT1	Option 1
25	MD4	MDATA Bit 4	26	TDMA	Terminate DMA
23	MD5	MDATA Bit 5	24		Reserved
21	MD6	MDATA Bit 6	22	MCS0/	M Chip Select 0
19	MD7	MDATA Bit 7	20	MCS1/	M Chip Select 1
17	GND	Signal Ground	18	+ 5V	+ 5 Volts
15	IORD/	I/O Read Command	16	MWAIT/	M Wait
13	IOWRT/	I/O Write Command	14	MINTR0	M Interrupt 0
11	MA0	M Address 0	12	MINTR1	M Interrupt 1
9	MA1	M Address 1	10		Reserved
7	MA2	M Address 2	8	MPST/	iSBX MULTIMODULE Board Present
5	RESET	Reset	6	MCLK	M Clock
3	GND	Signal Ground	4	+ 5V	+ 5 Volts
1	+ 12V	+ 12 Volts	2	- 12V	- 12 Volts

All undefined pins are reserved for future use.

APPENDIX B

iSBX™ MULTIMODULE™ BOARD I/O AC SPECIFICATIONS

Symbol*	Parameter	Min (ns)	Max (ns)
t ₁	Address stable before read	50	—
t ₂	Address stable after read	30	—
t ₃	Read pulse width	300	—
t ₄ ⁽²⁾	Data valid from read	0	250
t ₅ ⁽²⁾	Data float after read	0	150
t ₆	Time between RD and/or WRT	—	Note 3
t ₇	CS stable before CMD	25	—
t ₈	CS stable after CMD	30	—
t ₉	Power up reset pulse width	50 ms	—
t ₁₀	Address stable before WRT	50	—
t ₁₁	Address stable after WRT	30	—
t ₁₂ ⁽²⁾	Write pulse width	300	—
t ₁₃ ⁽²⁾	Data valid to write	250	—
t ₁₄	Data valid after write	30	—
t ₁₅	MCLK cycle	100	110
t ₁₆	MCLK width	35	65
t ₁₇ ⁽¹⁾	MWAIT/ pulse width	0	4 ms
t ₁₈	Reset pulse width	50 ms	—
t ₁₉	MCS/ to MWAIT/ valid	0	75
t ₂₀	DACK set up to I/O CMD	100	—
t ₂₁	DACK hold	30	—
t ₂₂	CMD to DMA RQT removed to end of DMA cycle	—	200
t ₂₃	TDMA pulse width	500	—
t ₂₄ ⁽¹⁾	MWAIT/ to valid read data	—	0
t ₂₅ ⁽¹⁾	MWAIT/ to WRT CMD	0	—

NOTES:

1. Required only if WAIT is activated.
 2. If MWAIT/ not activated.
 3. To be specified by each iSBX MULTIMODULE board.
- * For a more complete definition of symbols refer to iSBX Bus Specification, 142686-001.

APPENDIX C LISTING FOR THE iSBX™ DESIGN EXAMPLE SOFTWARE EXERCISER

LOC	OBJ	LINE	SOURCE STATEMENT
		1	*****
		2	;
		3	;* THIS PROGRAM WAS USED AS AN EXAMPLE FOR EXERCISING THE *
		4	;* 8279 iSBX MULTIMODULE BUILT FOR THIS APPLICATION NOTE. *
		5	;
		6	*****
		7	
		8	
		9	*****
		10	PROGRAM EQUATES
		11	*****
00F0		13	DATAAD EQU 0F0H ; PORT ADDRESS TO READ OR WRITE
		14	;/DATA TO/OR FROM KEYBOARD/DISPLAY
00F1		15	CMDAD EQU 0F1H ; PORT ADDRESS TO SEND COMMANDS
		16	;/TO KEYBOARD/DISPLAY
0008		17	MODE0 EQU 08H ; CONTROL CHAR. TO SET
		18	;/KEYBOARD/DISPLAY MODE FOR
		19	;/2 KEY LOCKOUT,16 CHAR LEFT ENTRY
0039		20	PROGCK EQU 39H ; CONTROL CHAR. TO SET 8279 CLK
		21	;/TO 100 KHZ INTERNAL TIMING
0040		22	RDFIFO EQU 40H ; CONTROL CHAR. TO READ KEYBOARD
0060		23	RDRAM EQU 60H ; CONTROL CHAR. TO READ DISPLAY RAM
0070		24	RDRAMA EQU 70H ; CONTROL CHAR. TO READ DISPLAY RAM
		25	;/AUTO INCREMENT
0080		26	WRRAM EQU 80H ; CNTL CHAR. TO WRITE TO DISPLAY RAM
0090		27	WRRAMA EQU 90H ; CNTL CHAR. TO WRITE TO DISPLAY
		28	;/RAM AUTO INCREMENT
00D8		29	CLR EQU 0D8H ; CONTROL CHAR. TO CLEAR OR BLANK
		30	;/DISPLAY
3C00		31	BUFF EQU 3C00H ; ADDRESS OF KEYBOARD INPUT BUFFER
3D00		32	DBUFF EQU 3D00H ; ADDRESS OF DISPLAY BUFFER
		33	
		34	*****
		35	
0000 F3		36	START: DI
0001 C33B00		37	JMP BEGIN
		38	
		39	***** RST 7 ENTRY POINT *****
		40	
0038		41	ORG 38H
0038 C3D100		42	JMP INT
		43	
		44	*****
		45	INITIALIZE PROGRAM
		46	AND KEY BOARD DISPLAY CONTROLLER
		47	;
003B 31FF3F		48	BEGIN: LXI SP,3FFFH ; INITIALIZE STACK PT
003E 3E08		49	MVI A,MODE0 ; GET CONTROL CHAR.
0040 D3F1		50	OUT CMDAD ; SET KEYBOARD/DISPLAY MODE
0042 3E39		51	MVI A,PROGCK ; GET CONTROL CHAR.
0044 D3F1		52	OUT CMDAD ; SET 8279 CLK FOR 100 KHZ
0046 3E08		53	MVI A,CLR ; GET CONTROL CHAR.
0048 D3F1		54	OUT CMDAD ; CLEAR OR BLANK DISPLAY
004A 0EE0		55	MVI C,0EH
004C 21003C		56	LXI H,BUFF ; SET POINTER TO INPUT BUFFER
004F 71		57	MOV M,C ; CLEAR INPUT BUFFER TO BLANK CODE
0050 060F		58	MVI B,0FH ; SET COUNTER = 15
0052 210F3D		59	LXI H,DBUFF+0FH ; SET POINTER TO DBUFF +15
0055 71		60	ZDBUFF: MOV M,C ; CLEAR DISPLAY BUFFER TO
0056 2B		61	DCX H ; /DISPLAY BUFFER +15 TO CODE
0057 05		62	DCR B ; /FOR CLEARING OR BLANKING OUT
0058 C25500		63	JNZ ZDBUFF ; /THE DISPLAY
		64	
		65	*****
		66	THIS IS THE BACKGROUND PROGRAM
		67	WHICH LOOPS CHECKING FOR THE DATA PRESENT FLAG
		68	;
005B F3		69	CKFLAG: DI ; DISABLE INTERRUPTS
005C AF		70	XRA A ; /CLEAR A REG AND COMPARE WITH
005D B9		71	CMP C ; /C REG CHECKING FOR DATA PRESENT
005E CA6400		72	JZ LABEL ; /IF PRESENT CALL OUTPT
0061 CD6800		73	CALL OUTPT ; /TO DISPLAY CHAR.
0064 FB		74	LABEL: EI ; /IF NO DATA PRESENT ENABLE
0065 C35800		75	JMP CKFLAG ; /INTERRUPTS AND JMP BACK
		76	

```

*****
77 ;*****
78 ;          OUTPUT CHARACTER TO DISPLAY
79 ;*****
0068 3A003C      80 OUTPT:      LDA      BUFF          ; LOAD A WITH KEYBOARD DATA
0068 062B        81          MVI      B,2BH          ; SET COUNTER MAX POSSIBLE CHAR.
006D 21DE00      82          LXI      H,TABLE1        ; SET POINTER TO INPUT TABLE
0070 110901      83          LXI      D,TABLE2        ; SET POINTER TO OUTPUT TABLE
0073 BE         84 COMPARE:    CMP      M          ; COMPARE KEYBD DATA TO INPUT
0074 CA8000      85          JZ       MATCH          ;/TABLE IF = JMP TO MATCH
0077 05         86          DCR      B          ;/ELSE DECREMENT COUNTER IF 0
0078 CAC600      87          JZ       CONTROL        ;/JMP TO CONTROL
007B 23         88          INX      H          ;/ELSE INCREMENT BOTH TABLE
007C 13         89          INX      D          ;/POINTERS AND JMP TO COMPARE
007D C37300      90          JMP      COMPARE
0080 EB         91 MATCH:     XCHG          ; IF MATCH CHANGE INPUT WITH
0081 7E         92          MOV      A,M          ;/OUTPT DATA AND PLACE IN BUFF
0082 21003C      93          LXI      H,BUFF
0085 77         94          MOV      M,A
0086 060F        95          MVI      B,0FH          ; SET COUNTER = TO 15
0088 11003D      96          LXI      D,DBUFF          ; POINTER TO FIRST LOC IN DBUFF
008B 21013D      97          LXI      H,DBUFF+1        ; POINTER TO 2ND LOC IN DBUFF
008E 7E         98 LOOP1:     MOV      A,M          ; READ HIGH POINTER FROM DBUFF
008F 23         99          INX      H          ;/UPDATE HIGH POINTER
0090 EB        100          XCHG
0091 77        101          MOV      M,A          ; SHIFT DATA LEFT IN D BUFF
0092 23        102          INX      H          ; UPDATE LOW POINTER
0093 EB        103          XCHG
0094 05        104          DCR      B          ; TEST IF DONE
0095 C28E00      105          JNZ      LOOP1          ;/AND GO BACK IF NOT
0098 3A003C      106          LDA      BUFF          ;/ELSE READ KEYBOARD DATA
009B 320F3D      107          STA      DBUFF+0FH        ;/AND PLACE IT IN THE DBUFF
009E 0610       108 LOOPA:     MVI      B,10H          ; SET COUNTER = 16
00A0 21003D      109          LXI      H,DBUFF          ; SET POINTER = DBUFF 1ST POS.
00A3 7E       110 LOOP2:     MOV      A,M          ;/READ 1 BYTE FROM DBUFF
00A4 D3F0       111          OUT      DATAAD        ;/AND SENT IT TO DISPLAY
00A6 23       112          INX      H          ; UPDATE POINTER
00A7 05       113          DCR      B          ;/AND TEST IF DONE
00A8 C2A300      114          JNZ      LOOP2          ;/GO BACK IF NOT DONE
00AB 0E00       115          MVI      C,0H          ;/ELSE CLR DATA PRESENT FLAG
00AD C9       116          RET          ;/AND RETURN
117 ;*****
118 ;          CHARACTER DFLETE
119 ;          OR RUB OUT
120 ;
00AE 060F       121 DELETE:    MVI      B,0FH          ; SET COUNTER =15
00B0 110F3D      122          LXI      D,DBUFF+0FH        ; SET POINTER = DBUFF+15
00B3 210E3D      123          LXI      H,DBUFF+0EH        ; SET POINTER = DBUFF+14
00B6 7E       124 LOOPB:     MOV      A,M          ; READ LOW POINTER FROM DBUFF
00B7 2B       125          DCR      H          ;/UPDATE LOW POINTER
00B8 EB       126          XCHG
00B9 77       127          MOV      M,A          ; SHIFT DATA RIGHT IN DBUFF
00BA 2B       128          DCR      H          ;/UPDATE HIGH POINTER
00BB EB       129          XCHG
00BC 05       130          DCR      B          ; TEST IF DONE
00BD C2B600      131          JNZ      LOOPB          ;/AND GO BACK IF NOT
00C0 EB       132          XCHG          ;/ELSE SET DBUFF FOR
00C1 36E0       133          MVI      M,0E0H          ;/CODE TO BLANK DISPLAY
00C3 C39E00      134          JMP      LOOPA          ;/AND JMP TO LOOPA
135
136 ;*****
137 ;          CHECK IF CHARACTER IS
138 ;          A CONTROL OR DELETE CHARACTER
139 ;
00C6 FEFA       140 CONTROL:    CPI      OFAH          ; COMPARE FOR CONTROL CHAR.
00C8 CA3B00      141          JZ       BEGIN          ;/IF CONTROL JMP TO BEGIN
00CB FEF9       142          CPI      OF9H          ;/ELSE COMP. FOR DELETE CHAR.
00CD CAAE00      143          JZ       DELETE        ;/IF DELETE JMP TO DELETE
00D0 C9       144          RET          ;/ELSE RETURN
145
146 ;*****
147 ;          KEYBOARD INPUT
148 ;          INTERRUPT ROUTINE
149 ;
00D1 3E40       150 INT:      MVI      A,RDFIFO        ; GET CONTL CHAR. TO READ FIFO
00D3 D3F1       151          OUT      CMDAD          ; SET 8279 FOR READ MODE
00D5 DBF0       152          IN       DATAAD        ; READ KEYBOARD DATA IN
00D7 21003C      153          LXI      H,BUFF          ; SET POINTER TO BUFF
00DA 77       154          MOV      M,A          ;/AND STORE KEYBOARD DATA
00DB 0EFF       155          MVI      C,0FFH          ;/THEN SET DATA PRESENT FLAG
00DD C9       156          RET          ;/AND RETURN
157

```


158 ;	*****	TABLE 1		
159 ;		ACCEPTABLE INPUT CHARACTERS FROM KEYBOARD		
160 ;				
161 ;				
00DE DE	162 TABLE1:	DB	0DEH,0FFH,0EFH,0EEH,0E5H,0F6H,0FEH,0C6H	
00DF FF				
00E0 EF				
00E1 EE				
00E2 E5				
00E3 F6				
00E4 FE				
00E5 C6				
00E6 C9	163	DB	0C9H,0CAH,0D2H,0DAH,0D3H,0C7H,0D1H,0D9H	
00E7 CA				
00E8 D2				
00E9 DA				
00EA D3				
00EB C7				
00EC D1				
00ED D9				
00EE D5	164	DB	0D5H,0EDH,0E6H,0F5H,0C1H,0F7H,0DDH,0E7H	
00EF ED				
00F0 E6				
00F1 F5				
00F2 C1				
00F3 F7				
00F4 DD				
00F5 E7				
00F6 FD	165	DB	0FDH,0DFH,0CCH,0D4H,0DCH,0E4H,0ECH,0F4H	
00F7 DF				
00F8 CC				
00F9 DC				
00FA C4				
00FB E4				
00FC EC				
00FD F4				
00FE FC	166	DB	0FCH,0C0H,0C8H,0D0H,098H,0A2H,0CFH,0AAH	
00FF C0				
0100 C8				
0101 D0				
0102 98				
0103 A2				
0104 CF				
0105 AA				
0106 EB	167	DB	0EBH,0E3H,0D8H	
0107 E3				
0108 D8	168	DB		

AP-96

```

169 ;*****
170 ;          TABLE 2
171 ;    ACCEPTABLE OUTPUT CHARACTERS TO DISPLAY
172 ;
173 TABLE2:  DB      0C1H,0C2H,0C3H,0C4H,0C5H,0C6H,0C7H,0C8H

0109 C1
010A C2
010B C3
010C C4
010D C5
010E C6
010F C7
0110 C8
0111 C9
0112 CA
0113 CB
0114 CC
0115 CD
0116 CE
0117 CF
0118 D0
0119 D1
011A D2
011B D3
011C D4
011D D5
011E D6
011F D7
0120 D8
0121 D9
0122 DA
0123 F1
0124 F2
0125 F3
0126 F4
0127 F5
0128 F6
0129 F7
012A F8
012B F9
012C F0
012D FD
012E EB
012F E0
0130 EA
0131 EF
0132 EE
0133 2D
0000

174          DB      0C9H,0CAH,0CBH,0CCH,0CDH,0CEH,0CFH,0D0H

175          DB      0D1H,0D2H,0D3H,0D4H,0D5H,0D6H,0D7H,0D8H

176          DB      0D9H,0DAH,0F1H,0F2H,0F3H,0F4H,0F5H,0F6H

177          DB      0F7H,0F8H,0F9H,0F0H,0FDH,0EBH,0E0H,0EAH

178          DB      0EFH,0EEH,02DH

179          END      START

```

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

BEGIN A 003B	BUFF A 3C00	CKFLAG A 005B	CLR A 00D8	CMDAD A 00F1	COMPAR A 0073	CONTR0 A 00C6
DATRAD A 00F0	DBUFF A 3D00	DELETE A 00AE	INT A 00D1	LABEL A 0064	LOOP1 A 009E	LOOP2 A 00A3
LOOPA A 009E	LOOPB A 0086	WATCH A 0080	MODE0 A 0008	OUTPT A 0068	PROGCK A 0039	RDFIFO A 0040
RDRAM A 0060	RDRAMA A 0070	START A 0000	TABLE1 A 00DE	TABLE2 A 0109	WRRAM A 0080	WRRAMA A 0090
ZDBUFF A 0055						

ASSEMBLY COMPLETE, NO ERRORS

*iCS Industrial Control
Series and Analog I/O
Expansion*

10

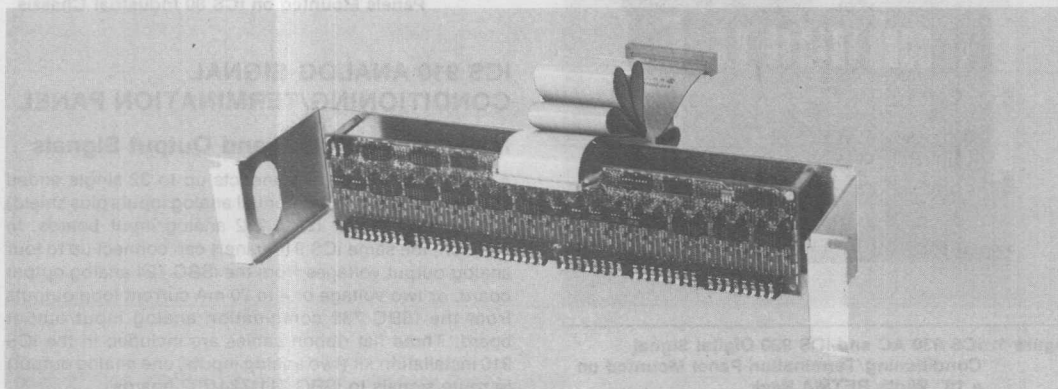


ICS 910/920/930

SIGNAL CONDITIONING/TERMINATION PANELS

- Interconnects iSBC and digital I/O ports to field signal/control wiring
- Ribbon cable connection from panel is pin compatible with iSBC analog, CPU, and digital board I/O ports
- Barrier strip screw terminals for
 - ICS 910: 32 single-ended analog inputs (or 16 differential signal plus shield) plus four analog voltage outputs or two analog 4 to 20 mA current outputs
 - ICS 920: 24 medium power digital inputs and/or outputs (55V, 300 mA max)
 - ICS 930: 16 high power AC or DC digital inputs or outputs (280 VAC, 3A max)
- Flexible mounting kits for
 - 19" width RETMA rack
 - NEMA type backwall
 - ICS 80 Industrial Chassis
- Digital signal conditioning (ICS 920/930)
 - Sockets for optically isolated input filters and solid state output switches
 - Pad space for transient suppressors, current limiting resistors, and voltage dividers
 - Socketed fuse for overload protection (ICS 930)
 - LED/channel status indicators
- Engineering printed circuit mounting space for customer analog input components (ICS 910)
 - Noise filters
 - Current loop resistors
 - Open circuit detection resistors
 - Voltage divider resistors
 - Thermistor bias current
- Submitted for UL recognition

The ICS 910/920/930 Signal Conditioning/Termination Panels are heavy duty printed circuit boards with screw terminations which allow industrial customers to easily connect their heavier gauge field signal wiring to Intel's line of 8- and 16-bit single board computers, and iSBC analog and digital I/O boards. Flat ribbon cables connect the ICS 910 panels to any of the Intel iSBC 700 series analog input and output board pin-outs. Flat ribbon cables also connect the ICS 920 panels and ICS 930 panel to the 50-pin digital I/O ports (8255 or UPI) on Intel's single board computers and digital I/O boards. Power for opto-isolators or line drivers (+5 VDC) can be supplied via this cable from the iSBC boards. Jumpers and a screw terminal block are provided on the ICS 920/930 panels to allow an external supply of +5V power. A similar jumper/terminal block is provided on the ICS 910 panel to allow users to connect external +15V (or greater) compliance voltage for larger analog output loads.



FUNCTIONAL DESCRIPTION COMMON TO iCS 910/920/930

Large Wire or Spade Lug Connections

The barrier strip screw terminations on the iCS 910/920/930 panels provide familiar connection points for factory electricians to terminate the heavier gauge wiring often pulled through conduits from sensors or control elements. These screw terminals securely connect up to 14 AWG gauge wire size (16-gauge on iCS 910/920 panels). Alternately, spade lugs can be crimped on field wiring and inserted under the screw terminals.

Mounting Flexibility and Serviceability

The iCS 910/920/930 panels were designed to be physically separate from iSBC boards or the iCS 80 chassis to allow maximum mounting flexibility and ease of serviceability. The panels and field wiring can be mounted in one area of the cabinet where electricians have access. Flat ribbon cable can then be run to the area where control electronics technicians have access.

The iCS 910/920/930 panels may be mounted horizontally in a 19" standard width (RETMA) rack using a recessed mounting panel (see Figure 1). Alternately, the panels can be mounted on a cabinet wall (e.g., NEMA cabinet backwall) using standoffs provided (see Figure 2). Or, for the most compact packaging, users can mount up to two iCS 910/920/930 panels vertically, directly on the front of the iCS 80 chassis using standoffs and holes provided (see Figure 3).

A black metal labelling strip is provided with each iCS 910/920/930 panel. White, blank gummed labels are included so that users can custom identify each input or output channel. A clear plastic cover is provided to protect against inadvertent touching or damage to the screw terminals or customer mounted components.

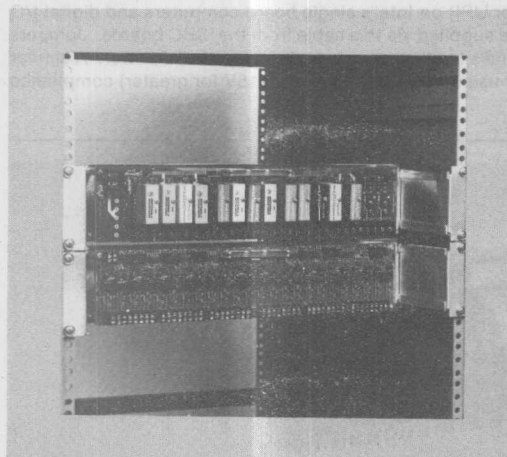


Figure 1. iCS 930 AC and iCS 920 Digital Signal Conditioning/Termination Panel Mounted on a 19" Width RETMA Rack

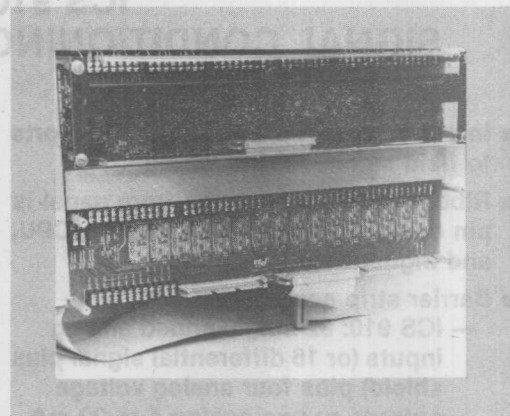


Figure 2. iCS 910/920 Signal Conditioning/Termination Panel Mounted on a NEMA Cabinet Backwall

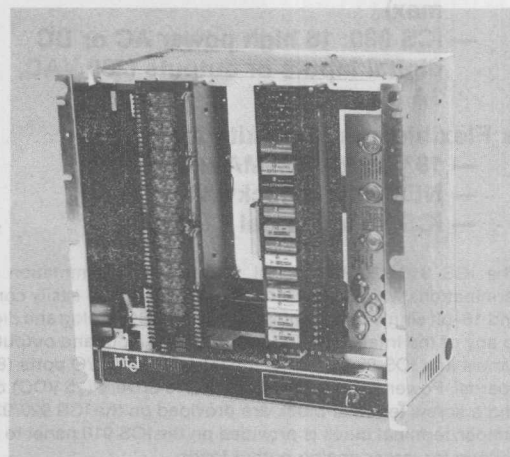


Figure 3. iCS 910/930 Signal Conditioning/Termination Panels Mounted on iCS 80 Industrial Chassis

iCS 910 ANALOG SIGNAL CONDITIONING/TERMINATION PANEL

Mixed Analog Input and Output Signals

A single iCS 910 panel connects up to 32 single ended analog inputs (or 16 differential analog inputs plus shield) to the iSBC 711 or iSBC 732 analog input boards. In addition, the same iCS 910 panels can connect up to four analog output voltages from the iSBC 724 analog output board, or two voltage or 4 to 20 mA current loop outputs from the iSBC 732 combination analog input/output board. Three flat ribbon cables are included in the iCS 910 installation kit (two analog inputs, one analog output) to route signals to iSBC 711/724/732 boards.

Engineered Signal Conditioning Mounting Space

Printed circuit traces on the iCS 910 panel connect each screw terminal analog input channel to the flat ribbon cable connector. Users can jump straight through signal connections if they desire. Each input channel trace, however, passes through a custom engineered printed

circuit area onto which users may mount components to signal condition analog input signals. Pad traces and holes are designed to allow easy mounting of R-C noise filters, input voltage resistor/divider networks, current loop input resistors, open circuit detection resistors, or to supply thermistor bias current (see Figure 4 for schematic of a typical analog input channel).

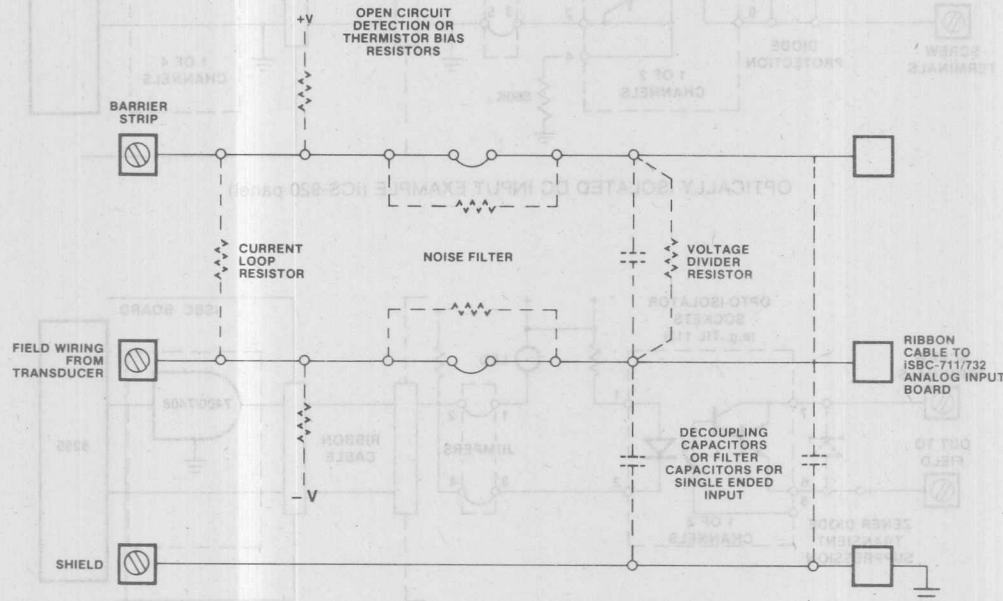


Figure 4. iCS 910 Analog Input Signal Conditioning Examples

ICS 920 DIGITAL SIGNAL CONDITIONING/TERMINATION PANEL

The iCS 920 panel interconnects up to 24, 2-wire digital input or output channels from barrier strip screw terminals to the 16- or 24-bit digital I/O ports, standard on many Intel single board computers and digital I/O expansion boards. Screw terminals allow for one each 16 AWG size wire for differential (2-wire) connections or two each AWG 18-gauge wire for daisy chaining grounds or power for external contact sensing.

Flexibility in Isolation and Serviceability

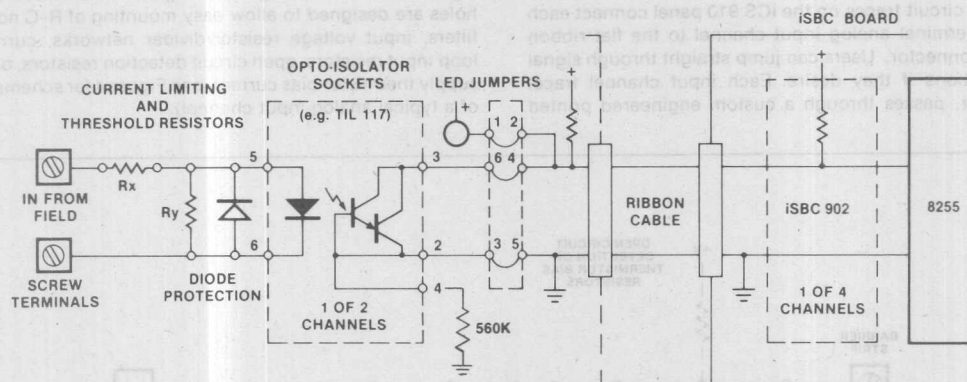
Dual-in-line sockets are in series with each channel (see Figure 5) to allow customer jumpering for straight through connections (TTL I/O), or for insertion of popular DIP packaged opto-isolators or digital output high current driver transistors. Circuit pads are available for

mounting voltage divider/threshold resistors and protection diodes.

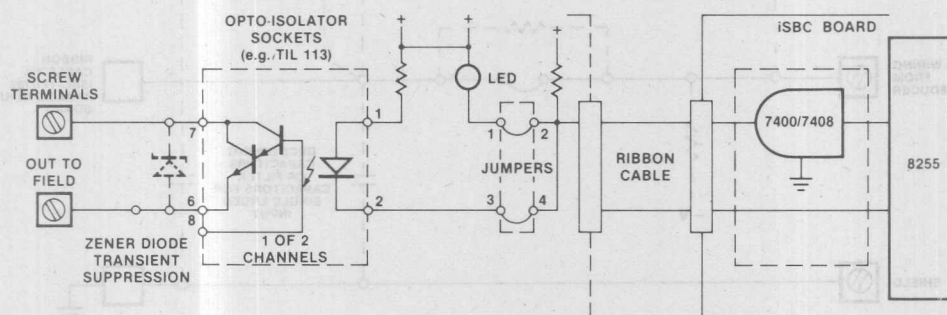
Groups of four inputs can have mixed voltage levels, opto-isolation, or straight through connections in groups of two. Output groups of four can be mixed opto-isolated or high current drive in groups of two. DIP components from a wide variety of vendors are selected and inserted by users based on their application. The iCS 920 manual recommends several alternative components and offers design assistance for your I/O configuration. Digital signal conditioning examples for several common industrial voltages are shown in Table 1 and in the diagrams below (see Figure 5).

Active Channel Indicators

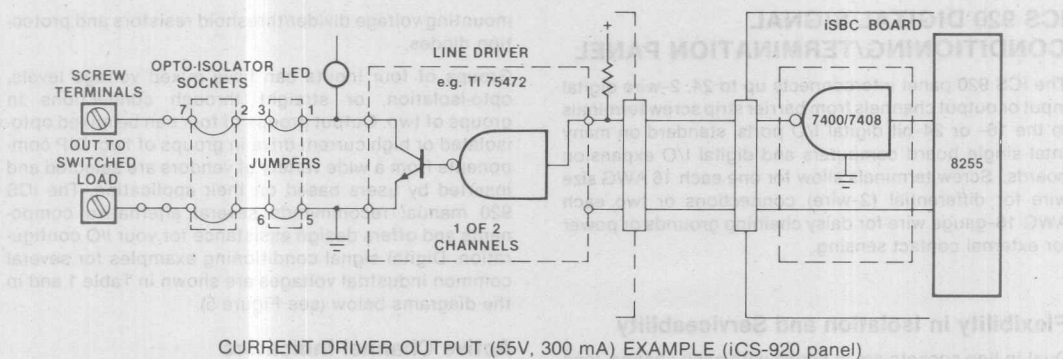
Light emitting diodes (LEDs) are mounted adjacent to each channel's screw terminals and may be jumpered in to indicate the Hi-Lo status of each of the 24 input or output channels.



OPTICALLY ISOLATED DC INPUT EXAMPLE (ICS-920 panel)



OPTICALLY ISOLATED DC OUTPUT EXAMPLE (ICS-920 panel)



CURRENT DRIVER OUTPUT (55V, 300 mA) EXAMPLE (ICS-920 panel)

Figure 5. Digital Signal Conditioning Examples

Table 1. ICS 920 Digital I/O Signal Conditioning Plug-In Component Examples

Digital Voltage Input or Output Load Voltage	Maximum Input Current (mA)	Threshold Voltage (V)	Opto-Isolators*	Diode Protection*
Opto-Isolated Input				
5 VDC	50	3	TIL117	1N4002
12 VDC	50	6	TIL117	1N4002
24 to 26 VDC	40	6	TIL117	1N4002
48 VDC	20	12	4N36	1N4002
	Maximum Output Current (mA)	Line Driver*	Opto-Isolators*	
Opto-Isolated Output				
12 VDC	100	—	TIL113	
24 VDC	100	—	TIL119	
48 VDC	100	—	MCS 2	
Current Drivers				
55 VDC	300	TI75472	—	
Half Wave Rectifier Outputs				
24 VAC SCR	300	—	GE4N40	
115 VAC SCR	150	—	MCS 2	

*Example component — alternate source components are listed in the ICS 920 Hardware Reference Manual.

ICS 930 AC Signal Conditioning/Termination Panel

The ICS 930 panel interconnects 16 2-wire digital input or output channels from barrier strip screw terminals to 16 bits of the digital I/O ports available on many Intel single board computers and digital I/O expansion boards. The ICS 930 panel differs from the ICS 920 digital signal conditioning/termination panel in that the ICS 930 panel handles higher AC or DC voltages and currents (up to 280V, 3A), such as those found on many 115 VAC machines, motor starters, and industrial control panels. The ICS 930 panel is also recommended for optically isolated DC outputs greater than 100 mA.

The ICS 930 screw terminals accept up to 14 AWG size wire each for differential (2 wires per channel) connections, or two 14 AWG size wires for daisy chaining grounds or power from external sources.

Modular Isolation/Switching with Easy Serviceability

Each ICS 930 panel accepts up to 16 user supplied, optically isolated input modules or optically isolated solid state switches, for either AC or DC voltages (see Figure 6). Each module is screw mountable/replaceable and can be mixed for AC or DC input, or AC or DC output, in groups of four. Among groups of four inputs (or outputs)

each channel can be individually mixed for AC or DC input (or AC or DC output). The user pays only for those channels implemented. User supplied compatible modules are shown in Table 2.

DC and AC input modules are current actuated and thus provide a 5-ms filter against spurious noise spikes or contact bounce. AC solid state output modules provide zero crossing turn on to minimize arcing.

Protection Circuitry

Each of the 16 channels contain a socketed fuse to protect against overload. In addition, mounting pads are available on each channel output for user supplied voltage transient RC "snubber" components or inductive pulse suppression, e.g., metallic-oxide-varistor (MOV) for large motor starting.

Active Channel Indicators

Light emitting diodes (LEDs) are mounted adjacent to each channel's screw terminals and opto-module to indicate Hi-Lo status of that channel and to assist in troubleshooting servicing.

Examples of ICS 930 input and output schematics are shown in Figure 6.

ICS 910/920/930

Table 2. Optically Isolated Modules Compatible with iCS 930 Signal Conditioning/Termination Panel

Signal Conditioning Desired	Voltage Rating	Maximum Input Current	Opto-22 Number*	Motorola Number*
AC Input — 115 VAC 220 VAC	95 to 130 VAC 180 to 280 VAC	10 mA 10 mA	IAC5 IAC5A	IAC5
DC Input — 5 μ sec Filter Fast, 50 μ sec On	10 to 32 VDC 4 to 16 VDC	32 mA 14 mA	IDC5 IDC5B	IDC5
Output Current Rating				
AC Output	12 to 140 VAC 24 to 280 VAC	3A 3A	OAC5 OAC5A	OAC5
DC Output	10 to 60 VDC 200 VDC	3A 1A	ODC5 ODC5A	ODC5

*Motorola and Opto-22 sales offices are located in North America, Europe, and Japan.

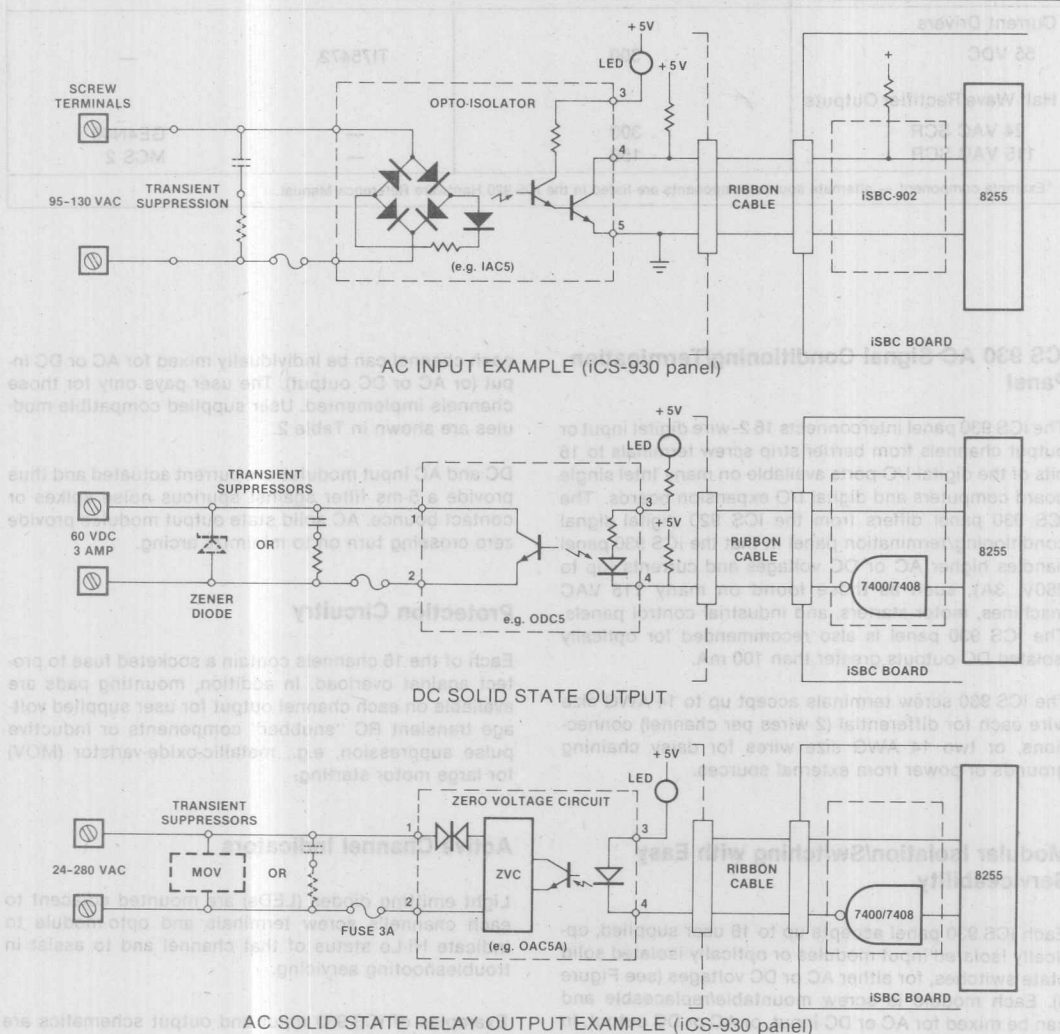


Figure 6. Typical iCS 930 Signal Conditioning Examples

ICS 910/920/930

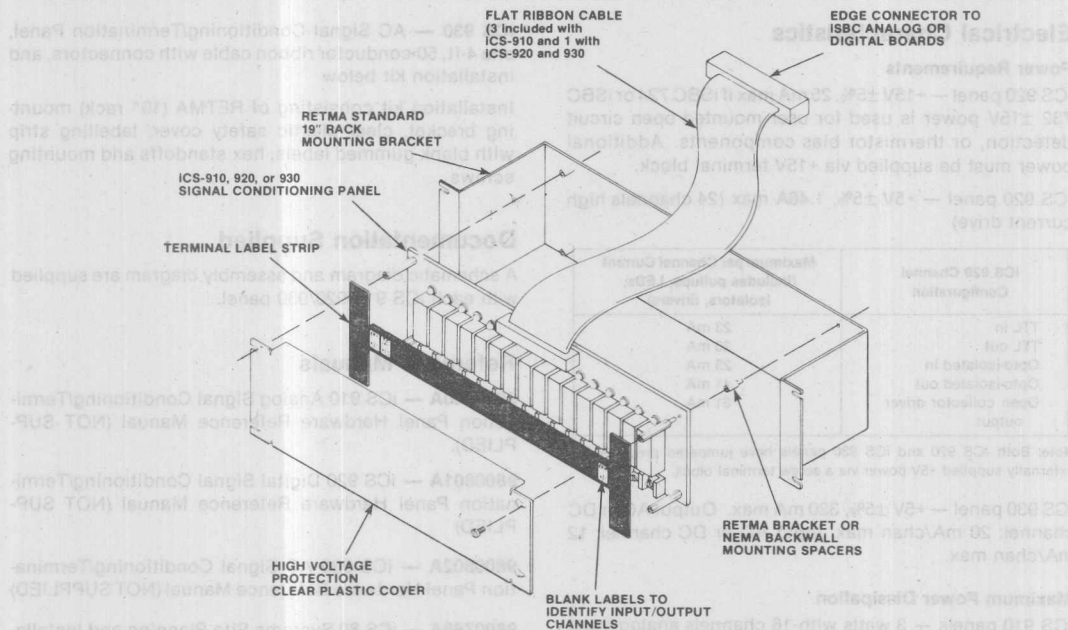


Figure 7. Mounting Arrangements for Signal Conditioning/Terminal Panels

SPECIFICATIONS

(For iCS 910/920/930 panels unless otherwise specified)

Number of Lines

iCS 910 Panel

Analog Inputs — Sixteen 3-wire (differential signal plus shield) or 32 single ended

Analog Outputs — Four 2-wire voltage output (iSBC 724 Analog Board) or two 2-wire current output (iSBC 732 Analog Board)

iCS 920 Panel — Zero to 24 digital inputs or outputs in groups of four

iCS 930 Panel — Zero to 16 digital inputs or outputs in groups of four

Isolation Characteristics

Line-to-Line Isolation — 250 VDC or RMS AC (iCS 910/920 panels), 500 VDC or RMS AC (iCS 930 panel)

Input/Output Isolation — 250 VDC or RMS AC (iCS 920 panel), 500 VDC or RMS AC (iCS 930 panel)

Physical Characteristics

Width: 36.63 cm (14.65 in.)

Height: 8.13 cm (3.25 in.)

Thickness: 0.24 cm (0.093 in.), iCS 910/920 panel
0.32 cm (0.125 in.), iCS 930 panel

iCS 910	iCS 920	iCS 930
Weight: (Minimum, PC panel only)		
455 gm (16 oz)	455 gm (16 oz)	681 gm (24 oz)
(Maximum with all components and mounting kit installed)		
1.6 Kg (56 oz)	1.8 Kg (64 oz)	3.4 Kg (120 oz)
Depth: (With components and clear plastic cover installed)		
5.08 cm (2.0 in.)	5.08 cm (2.0 in.)	5.08 cm (2.0 in.)
Connectors: (Barrier strip)		
2/56 screws	2/56 screws	6/32 screws
48 AI	48 DI/DO	32 DI/DO
12 AO	2 + 5V power	2 + 5V power
2 power	(J1, J2, J3 to iSBC boards)	
50-pin	50-pin	50-pin
0.1 in. centers	0.1 in. centers	0.1 in. centers
(2.54 mm)	(2.54 mm)	(2.54 mm)
(Mating connector: 3M 3415-0000 or TI H3-12125)		

Maximum Distance from iSBC Boards

The iCS 910/920/930 panels are shipped with 4-ft. long cables. With customer provided 50-conductor or twisted pair ribbon cable, however, the iCS 910/920/930 panels can be mounted remote from the iSBC analog or digital I/O boards. In electrically quiet environments using normal iSBC board line driver/receivers, the iCS 910/920/930 panels should be able to operate up to 25 ft. (7.69m) from the iSBC board.

Electrical Characteristics

Power Requirements

ICS 920 panel — +15V \pm 5%, 25 mA max if iSBC 724 or iSBC 732 \pm 15V power is used for user mounted open circuit detection, or thermistor bias components. Additional power must be supplied via +15V terminal block.

ICS 920 panel — +5V \pm 5%, 1.46A max (24 channels high current drive)

ICS 920 Channel Configuration	Maximum per Channel Current (Includes pullups, LEDs, isolators, drivers)
TTL in	23 mA
TTL out	23 mA
Opto-isolated in	23 mA
Opto-isolated out	41 mA
Open collector driver output	61 mA

Note: Both ICS 920 and ICS 930 panels have jumpered provision for externally supplied +5V power via a screw terminal block.

ICS 930 panel — +5V \pm 5%, 320 mA max. Output AC or DC channel: 20 mA/chan max; Input AC or DC channel: 12 mA/chan max.

Maximum Power Dissipation

ICS 910 panels — 3 watts with 16 channels analog input signal conditioning and +15V external compliance voltage

ICS 920 panels — 12 watts with 24 channels each containing high current driver outputs

ICS 930 panels — 80 watts with 16 channels of AC or DC output

Underwriters Laboratory (UL) Recognition

The ICS 910/920/930 signal conditioning/termination panels have been submitted to Underwriters Laboratories for approval as a UL recognized component under the UL safety standard for process control equipment, UL 1092.

Environmental Characteristics

Operating Temperature — 0 to 70°C (32°F to 158°F)

Relative Humidity — 0 to 90%, noncondensing

Hardware Supplied

ICS 910 — Analog Signal Conditioning/Terminating Panel, three 4-ft, 50-conductor flat ribbon cables with connectors, and installation kit below

ICS 920 — Digital Signal Conditioning/Termination Panel, one 4-ft, 50-conductor flat ribbon cable with connectors, and installation kit below

ICS 930 — AC Signal Conditioning/Termination Panel, one 4-ft, 50-conductor ribbon cable with connectors, and installation kit below

Installation kit consisting of RETMA (19" rack) mounting bracket, clear plastic safety cover, labelling strip with blank gummed labels, hex standoffs and mounting screws

Documentation Supplied

A schematic diagram and assembly diagram are supplied with each ICS 910/920/930 panel.

Reference Manuals

9800800A — ICS 910 Analog Signal Conditioning/Termination Panel Hardware Reference Manual (NOT SUPPLIED).

9800801A — ICS 920 Digital Signal Conditioning/Termination Panel Hardware Reference Manual (NOT SUPPLIED)

9800802A — ICS 930 AC Signal Conditioning/Termination Panel Hardware Reference Manual (NOT SUPPLIED)

9800798A — ICS 80 Systems Site Planning and Installation Guide (NOT SUPPLIED), but supplied with ICS 80 Industrial Chassis

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

Installation

Complete instructions for installation and service are contained in the applicable ICS 910/920/930 Hardware Reference Manual. Additional system level information is available in the ICS 80 Systems Site Planning and Installation Guide, including RETMA and NEMA cabinet mounting, field signals, ground wiring and cooling suggestions.

Warranty

The ICS 80 Industrial Chassis is warranted to be free from defects in materials and workmanship under normal use and service for a period of 90 days from date of shipment.

ORDERING INFORMATION

Part Number Description

ICS 910 Analog signal conditioning/termination panel

ICS 920 Digital signal conditioning/termination panel

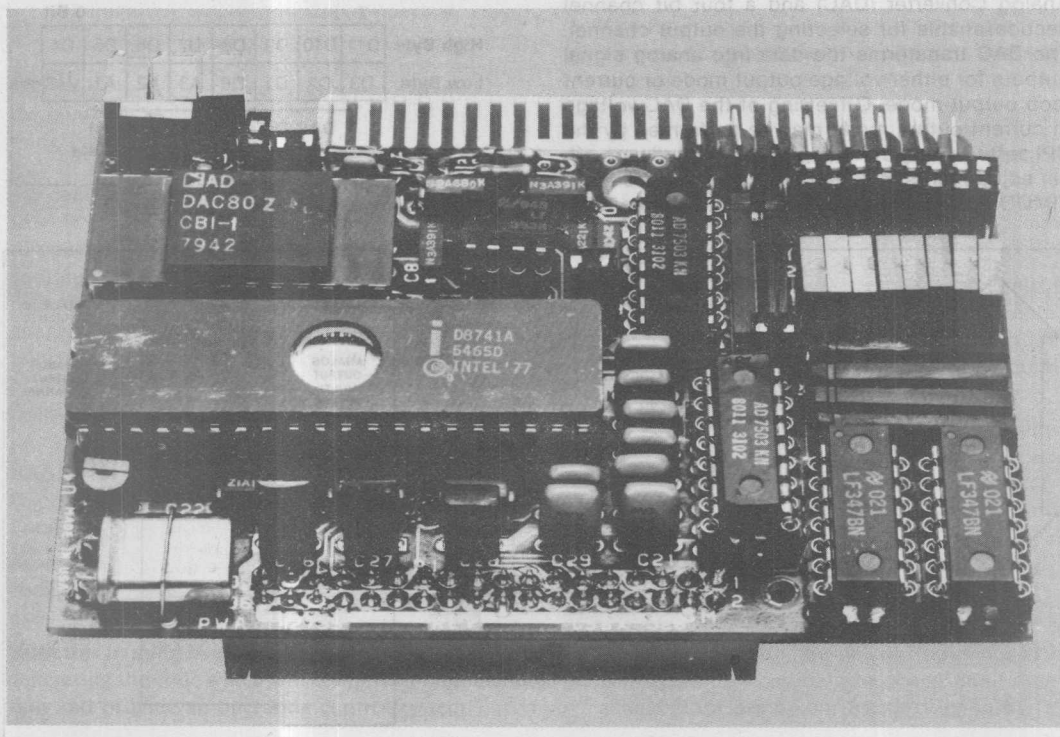
ICS 930 AC signal conditioning/termination panel



iSBX 328 ANALOG OUTPUT MULTIMODULE EXPANSION BOARD

- Low cost analog output for iSBX MULTIMODULE compatible iSBC Boards
- 8 channels output, current loop or voltage in any mix
- 4-20 mA current loop; 5V unipolar or bipolar voltage output
- 12-bit resolution
- 0.035% full scale volage accuracy @ 25°C
- Connector compatible with ICS 910 Analog Termination Panel
- Intel design based on UPI control for high density and low cost
- Programmable offset adjust in current loop mode

The Intel iSBX 328 MULTIMODULE board provides analog signal output for any iSBC board which has an iSBX compatible bus and connectors. The single-wide iSBX 328 plugs directly onto the iSBC board, providing eight independent output channels of analog voltage for meters, CRT control, programmable power supplies, etc. Voltage output can be mixed with current loop output for control of popular 4-20ma industrial control elements. By using an Intel single chip computer LSI (8041) for refreshing separate sample-hold amplifiers through a single 12 bit DAC, eight channels can be contained on a single MULTIMODULE board, for high density and low cost per channel. High quality analog components provide 12 bit resolution, 11 bit accuracy, and slew rates per channel of 0.1 volt per microsecond. Programming the iSBX 328 MULTIMODULE board is done via a simple two byte protocol over the iSBX bus. Maximum channel update rates are 5KHZ on a single channel to 1 KHZ on all eight channels. Outputs are compatible for screw termination of field wiring on the iCS 910 Analog Signal Conditioning/Termination Panel.



ANALOG OUTPUT MULTIMODULE
EXPANSION BOARD

EXPANSION BOARD

Interfacing Through the Intel iSBX Bus

All data to be output through the MULTIMODULE board is transferred from the host iSBC micro-computer to the MULTIMODULE board via the iSBX bus connector. The UPI device on the MULTIMODULE board accepts the binary digital data and generates a 12-bit data word for the Digital-to-Analog Converter (DAC) and a four bit channel decode/enable for selecting the output channel. The DAC transforms the data into analog signal outputs for either voltage output mode or current loop output mode. Offsetting of the DAC voltage in current output mode may be performed by the UPI software offset routine or by the hardware offset adjustments included on the board. The MULTIMODULE board status is available via the iSBX

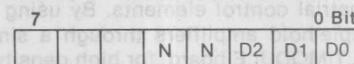
bus connector, to determine if the UPI is ready to receive updates to analog output channels.

OPERATIONAL DESCRIPTION

The host iSBC microcomputer addresses the MULTIMODULE board by executing IN or OUT instructions specifying the ISBX 328 MULTIMODULE as a port address. The UPI on the iSBX 328 is initialized to select whether software or hardware offset is to be used and how many channels will be active. Then a 2 byte transfer to each active channel sets the 12 bit output value, the channel selected and the current or voltage mode.

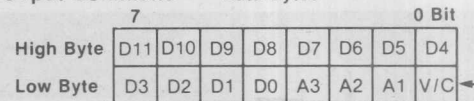
Commands

OUTput Command — Initialization of UPI/iSBX 328



NN: 0,0 = unipolar configuration
 software current offset
0,1 = no mixing
1,0 = bipolar configuration
 software current offset

OUTput Command — Data Bytes



0 = UPI generates offset	} to receive data
1 = SBC generates offset in current loop mode	

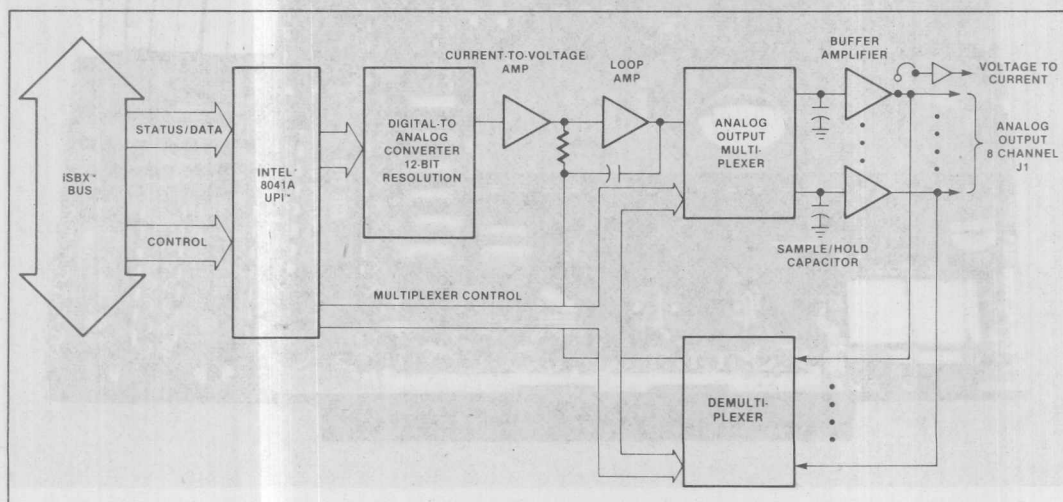
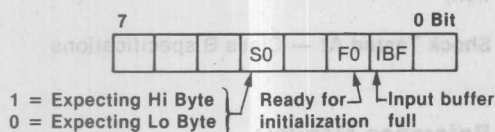


Figure 1. iSBC 328 Analog Output MULTIMODULE Board Block Diagram

ISBX 328

INput Command — Status Buffer Read



SPECIFICATIONS

Outputs—8 non-isolated channels, each independently jumpered for voltage output or current loop output mode.

Voltage Ranges—0 to +5 volts (unipolar operation)
–5 to +5 volts (bipolar operation)

Current Loop Range—4 to 20 mA (unipolar operation only)

Output Current—±5 mA maximum (voltage mode-bipolar operation)

Load Resistance—0 to 250 ohms with on-board iSBX power. 1000 ohms minimum with 30 VDC max. external supply

Compliance Voltage—12 V using on-board iSBX power. If supplied by user, up to 30 VDC max

Resolution—12 bits bipolar or unipolar

Slew Rate—0.1 volt per microsecond minimum

Single Channel
Update Rate—5KHz

Eight Channel
Update Rate—1KHz

Accuracy—

Mode	Accuracy	Ambient Temp
Voltage-Unipolar, typical	± 0.025% FSR	@ 25°C
Voltage-Unipolar, maximum	± 0.035% FSR	@ 25°C
Voltage-Unipolar, typical	± 0.08% FSR	@ 0° to 60°C
Voltage-Unipolar, maximum	± 0.19% FSR	@ 0° to 60°C
Voltage-Bipolar, typical	± 0.025% FSR	@ 25°C
Voltage-Bipolar, maximum	± 0.035% FSR	@ 25°C
Voltage-Bipolar, typical	± 0.09% FSR	@ 0° to 60°C
Voltage-Bipolar, maximum	± 0.17% FSR	@ 0° to 60°C
Current Loop, typical	± 0.07% FSR	@ 25°C
Current Loop, maximum	± 0.08% FSR	@ 25°C
Current Loop, typical	± 0.17% FSR	@ 0° to 60°C
Current Loop, maximum	± 0.37% FSR	@ 0° to 60°C

Interrupts

No interrupts are issued from the iSBX 328 to the host iSBC microcomputer. Data coordination is handled via iSBC software polls of the status buffer.

Refresh and Throughput Rates**

Refresh 1 channel (no new data):	80 us
Refresh all 8 channels (no new data):	650 us
Update and refresh 1 channel with new data: firmware program 2	150 us
for each additional channel	130 us
Update and refresh 1 channel with new data: firmware program 1 or 3	200 us
for each additional channel	155 us
Update and refresh all 8 channels (all new data): firmware program 2	1.050 ms
per channel of new data	50 us
Update and refresh all 8 channels (all new data): firmware program 1 or 3	1.280 ms
per channel of new data	80 us

** All times nominal

Output Impedance—0.1 ohm. Drives capacitive loads up to 0.05 microfarads. (approx. 1000 foot cable)

Temperature Coefficient—0.005%/°C

Connectors —

Interface	Pins (Qty)	Centers in	Centers cm	Mating Connectors
P1 iSBX Bus	36	0.1	0.254	iSBC iSBX connector
J1 8/16 channels analog	50	0.1	0.254	3m 3415-000 or T1 H312125 or iCS 910 cable

Physical Characteristics

Width — 9.40 cm (3.7 inches)

Length — 6.35 cm (2.5 inches)

Height — 1.4 cm (0.56 inch) MULTIMODULE board only
2.82 cm (1.13 inches) MULTIMODULE and iSBC board.

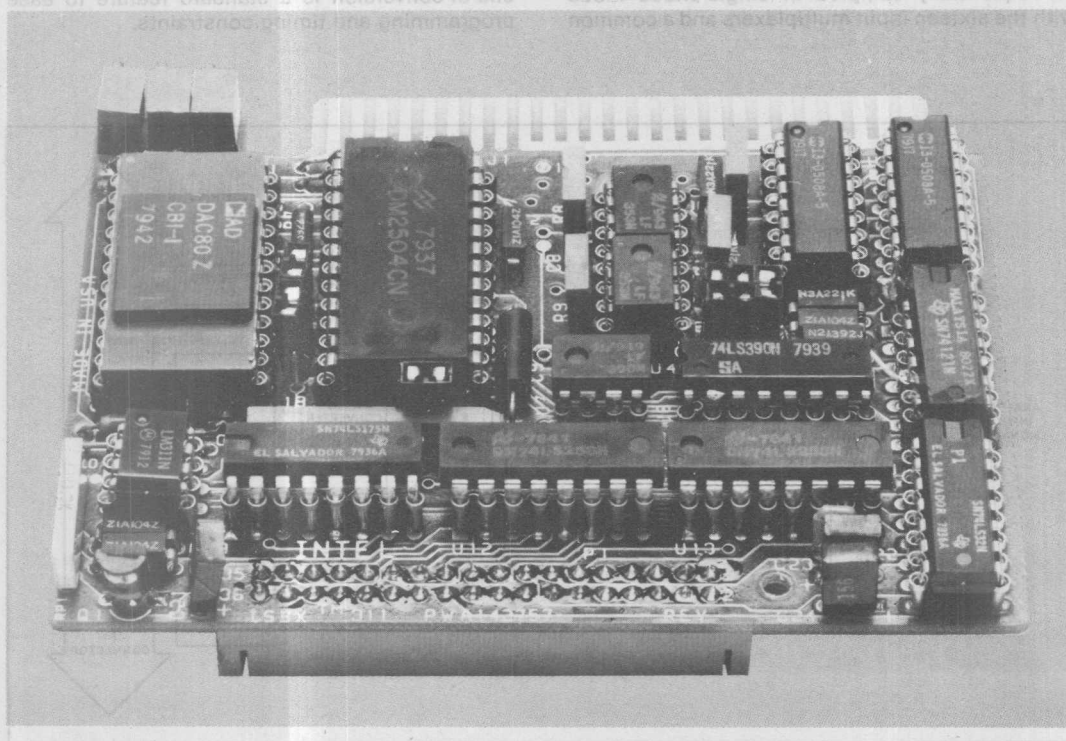
Weight — 85.06 gm (3.0 ounces)



iSBX 311 ANALOG INPUT MULTIMODULE BOARD

- Low cost analog input for iSBX MULTIMODULE compatible iSBC boards
- 8 differential/16 single-ended, fault protected inputs
- 20 mV to 5V full scale input range, resistor gain selectable
- Unipolar (0 to +5V) or bipolar (-5V to +5V) input, jumper selectable
- 12-bit resolution analog-to-digital converter
- 0.035% full scale accuracy (11 bits) at 25°C
- 18 kHz samples per second throughput to memory
- Connector compatible with iCS 910 Analog Termination Panel

The Intel iSBX 311 Analog Input MULTIMODULE board provides simple interfacing of non-isolated analog signals to any iSBC board which has an iSBX compatible bus and connectors. The single-wide iSBX 311 plugs directly onto the iSBC board, providing data acquisition of analog signals from eight differential or sixteen single-ended voltage inputs, jumper selectable. The iSBX 311 MULTIMODULE is connector and pinout compatible with the Intel iCS 910 Analog Signal Conditioning/Termination panel so that field wiring can easily be terminated and current loop-to-voltage conversion resistors can be mounted for current loop analog signal monitoring. Resistor gain selection is provided for both low level (20mv full scale range) and high level (5 volt FSR) signals. Incorporating the latest high quality IC components, the iSBX 311 MULTIMODULE board provides 12 bit resolution, 11 bit accuracy, and a simple programming interface, all on a low cost iSBX MULTIMODULE board.



FUNCTIONAL DESCRIPTION

The iSBX 311 Analog Input MULTIMODULE board is a member of Intel's growing family of MULTIMODULE expansion boards, designed to allow quick, easy, and inexpensive expansion for the Intel single board computer product line. The iSBX 311 Analog Input MULTIMODULE Board shown in figure 1, is designed to plug onto any host iSBC microcomputer that contains an iSBX bus connector (P1). The board provides 8 differential or 16 single-ended analog input channels that may be jumper-selected as the application requires. The MULTIMODULE board includes a user-configurable gain, and a user-selectable voltage input range (0 to +5 volts, or -5 to +5 volts). The MULTIMODULE board receives all power and control signals through the iSBX bus connector to initiate channel selection, sample and hold operation, and analog-to-digital conversion.

Input Capacity

Sixteen separate analog signals may be randomly or sequentially sampled in single-ended mode with the sixteen input multiplexers and a common

ground. For noisier environments, differential input mode can be configured to achieve 8 separate differential signal inputs, or 16 pseudo-differential inputs.

Resolution

The iSBX 311 MULTIMODULES provide 12-bit resolution with a successive approximation analog-to-digital converter. For bipolar operation (-5 to +5 volts) it provides 11 bits plus sign.

Speed

The A-to-D converter conversion speed is 35 microseconds (28KHZ samples per second). Combined with the sample and hold, settling times and the programming interface, maximum throughput via the iSBX bus and into memory will be 54 microseconds per sample, or 18 KHZ samples per second, for a single channel, a random channel, or a sequential channel scan. A-to-D conversion is initiated via the iSBX connector and programmed command from the iSBC base board. Interrupt on end-of-conversion is a standard feature to ease programming and timing constraints.

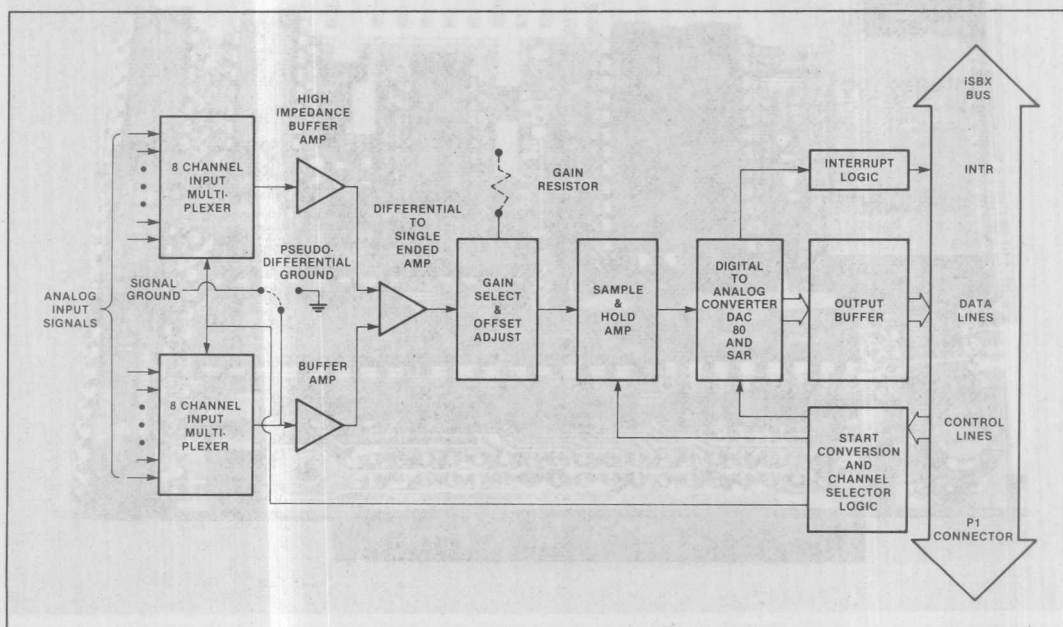


Figure 1. iSBX 311 Analog Input MULTIMODULE Board

Accuracy

High quality components are used to achieve 12 bits resolution and accuracy of .035% full scale range $\pm \frac{1}{2}$ LSB. Offset and gain are adjustable to $\pm 0.024\%$ FSR $\pm \frac{1}{2}$ LSB accuracy at any fixed temperature between 0°C (gain = 1). See specifications for other gain accuracies.

Gain

To allow sampling of millivolt level signals such as strain gauges and thermocouples, gain is made configurable via user inserted gain resistors up to $250 \times$ (20 millivolts, full scale input range). User can select any other gain range from 1 to 250 to match his application.

OPERATIONAL DESCRIPTION

The host iSBC microcomputer addresses the iSBX 311 MULTIMODULE board by executing IN or OUT instructions to the iSBX 311 MULTIMODULE as one of the legal port addresses. Analog-to-digital conversions can be programmed in either of two modes: 1. start conversion and poll for end-of-conversion (EOC), or 2. start conversion and wait for interrupt (INTRO/) at end of conversion. When conversion is complete as signaled by one of the above techniques, INput instructions read two bytes (low and high bytes) containing the 12 bit data word plus status information as shown below.

SPECIFICATIONS

Inputs — 8 differential. 16 single-ended. Jumper selectable.

Full Scale Input

Voltage Range — -5 to +5 volts (bipolar). 0 to +5 volts (unipolar). Jumper selectable.

Gain — User-configurable through installation of two resistors. Factory-configured for gain of X1; gains above 250 not recommended.

Resolution — 12 bits over full scale range (1.22 mv at 0-5 v, 5 μ v at 0-20 mv)

OUTput Command — Select input channel and start conversion.

Bit Position	7	6	5	4	3	2	1	0
Input Channel					C3	C2	C1	C0

Input Data — Read converted data and status (low byte) or Read converted data (high byte). Reads can be with or without reset of interrupt request line (INTRO/).

Bit Position	7	6	5	4	3	2	1	0
Low/status Byte	D3	D2	D1	D0		start/busy	EOC	

High Byte	D11	D10	D9	D8	D7	D6	D5	D4
-----------	-----	-----	----	----	----	----	----	----

Fastest data conversion and transfer to memory can be obtained by dedicating the microcomputer to setting the channel address/starting conversion, polling the status byte for EOC/, and when it comes true, read the two bytes of the conversion and send the start conversion/next channel address command. For multitasking situations it may be more convenient to use the interrupt mode, reading in data only after an interrupt signals end of conversion.

Accuracy —

Gain	Accuracy at 25°C
1	$\pm 0.035\% \pm \frac{1}{2}$ LSB
5	$\pm 0.035\% \pm \frac{1}{2}$ LSB
50	$\pm 0.035\% \pm \frac{1}{2}$ LSB
250	$\pm 0.035\% \pm \frac{1}{2}$ LSB

NOTE:

Figures are in percent of full scale reading. At any fixed temperature between 0° and 60°C, the accuracy is adjustable to $\pm 0.035\%$ of full scale.

Dynamic Error — $\pm 0.015\%$ FSR for transitions

Gain TC (at Gain = 1): 30 PPM per degree centigrade (typical); 56 PPM per degree centigrade (max).

Offset TC (in percent of FSR/°C):

Gain	Offset
1	.0018
5	.0036
50	.024
250	.116

Offset is measured with user-supplied 10 PPM/°C gain resistors installed.

Input Protection — ± 30 volts.

Input Impedance — 20 megohms (minimum).

Conversion Speed — 50 microseconds (nominal).

Common Mode Rejection Ratio — 60 db (minimum).

Sample and hold — sample time 15 microseconds.

Aperature — hold aperature time: 120 nanoseconds.

Connectors —

Interface	Pins (Qty)	Centers in	cm	Mating Connectors
P1 iSBX Bus	36	0.1	0.254	iSBC iSBX connector
J1 8/16 channels analog	50	0.1	0.254	3m 3415-000 or T1 H312125 or iCS 910 cable

ORDERING INFORMATION

Part Number	Description
SBX 311	Analog Input MULTIMODULE Board

NOTE:
Figures are in percent of full scale reading. At any fixed temperature between 0° and 60°C, the accuracy is adjustable to $\pm 0.003\%$ of full scale.

Dynamic Error — $\pm 0.01\%$ FSR for transitions

Gain TC (at Gain=1): 30 PPM per degree centigrade (typical); 50 PPM per degree centigrade (max).

Physical Characteristics

Width — 9.40 cm (3.7 inches)

Length — 6.35 cm (2.5 inches)

Height — 2.03 cm (0.80 inch) MULTIMODULE board only

2.82 cm (1.13 inches) MULTIMODULE and iSBC board

Weight — 68.05 gm (2.4 ounces)

Electrical Characteristics (from iSBX connector)

$V_{CC} = \pm 5$ volts ($\pm 0.25V$), $I_{CC} = 250$ mAmax

$V_{DD} = +12$ volts ($\pm 0.6V$), $I_{DD} = 50$ mAmax

$V_{SS} = -12$ volts ($\pm 0.6V$), $I_{SS} = 55$ mAmax

Environmental Characteristics

Operating Temperature — 0° to 60°C (32° to 140°C)

Relative Humidity — to 90% (without condensation)

Shock Tested At — Class B Specification

Reference Manuals

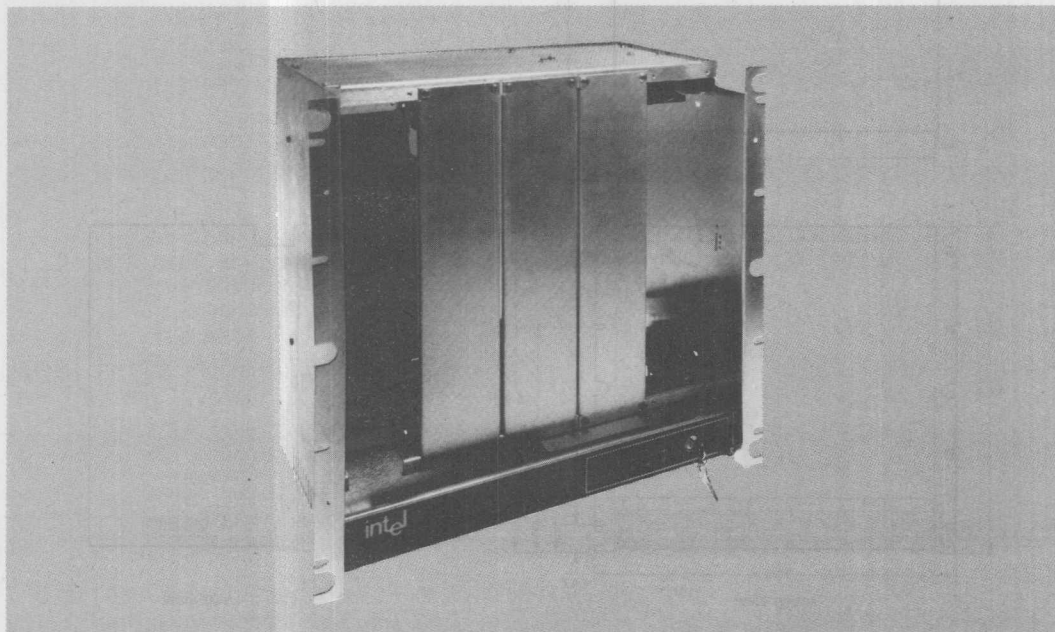
142913-001 — iSBX 311 Analog Input MULTIMODULE Board Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

INDUSTRIAL CHASSIS

- **MULTIBUS standard 4-slot backplane, expandable to 12 slots**
- **Vertical board orientation for convection cooling**
- **19-inch wide RETMA rack mounting or NEMA type backwall mounting brackets**
- **Four fans for forced-air cooling**
- **Submitted for approval as a UL recognized component**
- **110/230V, 50/60 Hz operation**
- **Slide in/out mounting rails for iSBC power supplies**
 - Quick disconnect cabling for serviceability
 - Your choice of supply
- **Lockable service panel**
- **All front access serviceability**
 - iSBC boards
 - Power supplies
 - Interrupt and reset buttons
 - Operation indicators and fuse
- **Recessed mounting space for signal conditioning/wire termination panels**

The iCS 80 Industrial Chassis provides industrially oriented mounting space for Intel single board computer (iSBC) products, associated iSBC power supplies, and related iCS 9XX analog and digital signal conditioning/termination panels. The base unit provides a 4-slot MULTIBUS backplane (iSBC 604) with expansion space and cabling to expand to 12 MULTIBUS backplane slots by adding additional 4-slot iSBC 614s as needed (up to two). All of the 25-plus Intel MULTIBUS bus-compatible iSBC boards can be inserted into any one of the 12 slots. In addition, over 50 products from 30 independent manufacturers have been designed for mounting into the MULTIBUS backplane. Full MULTIBUS compatibility in the iCS 80 chassis also allows configuration of multiple single board computers to share system tasks through communication over the bus (through multimaster bus arbitration built on the multiple iSBC processors).



FUNCTIONAL DESCRIPTION

Self Contained Low Cost Controllers

Small, self contained industrial controllers can be configured with the 4-slot cardcage and iSBC 635 power supply. As shown in Figure 2, this packaging can also accommodate the iCS 9XX series signal conditioning/termination panels.

Or Large Power and Point Counts in a Small Package

At the high end of performance for the iCS 80 chassis, a user can build a 12-slot configuration with the Intel iSBC 640 Power Supply. This iCS 80 chassis can support the iSBC 86/12A 16-bit computer with 112K bytes memory (96K RAM, 16K ROM), 64 differential analog inputs, 180 digital inputs, 52 isolated digital outputs, and 8 analog outputs (four current loops); in total a 304-channel, mixed analog and isolated digital, input and output controller, large enough for most dedicated applications (see Figure 3 for two other examples).

Engineered for Industrial Applications

The MULTIBUS slots are mounted vertically to improve convection cooling and the top, bottom and sides are engineered to allow maximum air flow over the boards. Four fans are provided as standard to increase air flow, allowing users to eliminate or minimize the need for supplementary fans or air conditioning.

Power Supply Flexibility

To provide a modular base on which to build a variety of configurations, no power supply is provided in the iCS 80 Industrial Chassis. Users choose one of the low cost Intel iSBC 635 (14-amp) or iSBC 640 (30-amp) power supplies based on their application. Slide in/out mounting rails are provided to match the iSBC 635 and iSBC 640 supplies, and quick disconnect cabling and connectors are provided for rapid service replacement. An AC wiring barrier strip allows simple wiring connections for integration into larger systems (see Figure 4).

Industrial Rack Mounting

The chassis mounts directly into 19-inch standard width RETMA (Radio-Electronics-Television Manufacturers Association) customer provided rack. Alternately, mounting brackets and power cabling access are provided for mounting directly on a backwall, such as the backwall panel of a NEMA-type (National Electrical Manufacturers Association), front-access-only cabinet.

Front Access Serviceability

To simplify serviceability, front access is provided for all iSBC boards, the power supply, operation indicator lights, interrupt and reset buttons, and the AC power fuse.

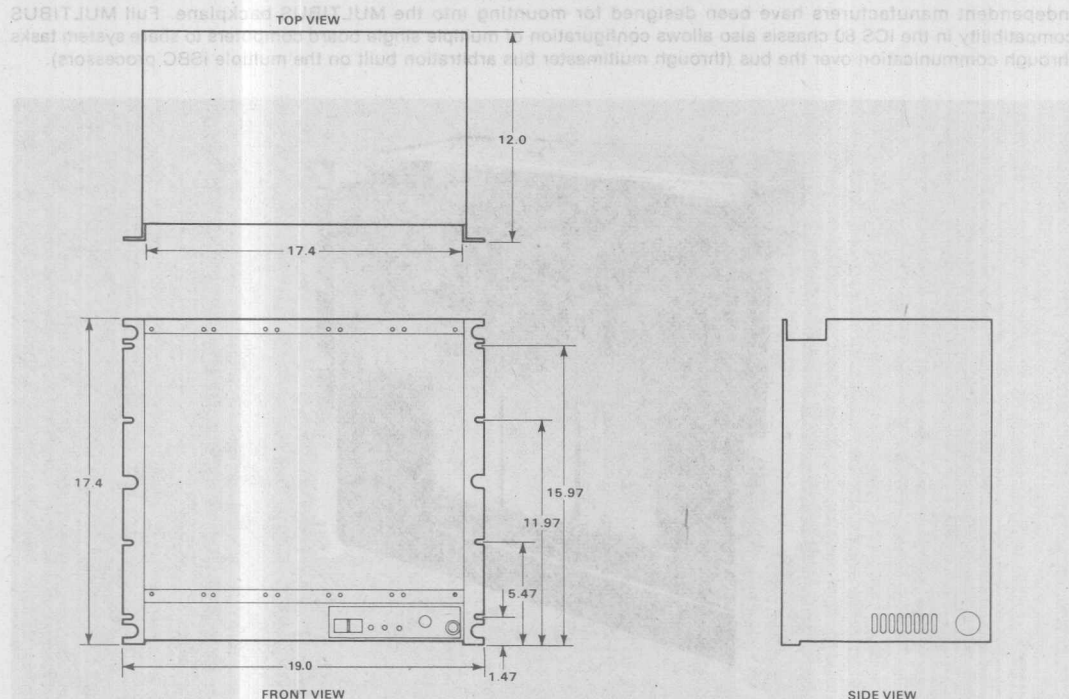


Figure 1. iCS 80 Chassis Dimensions

ICS 80

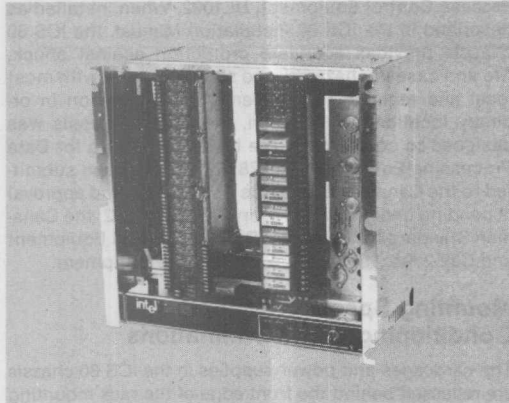


Figure 2. Small Configuration iCS 80 with iSBC 635, iCS 910 and iCS 930 Signal Conditioning/Termination Panels

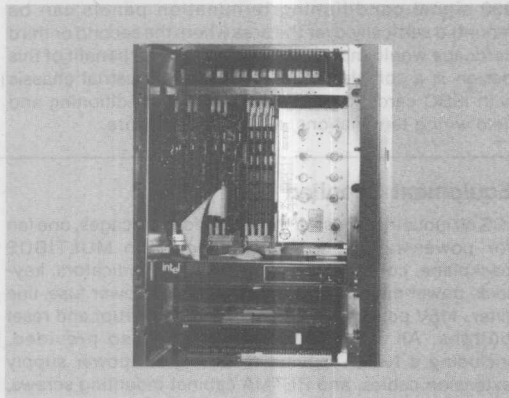


Figure 3. iCS 80 with 12 MULTIBUS Card Slots and iSBC 640 Power Supply, Large Configuration

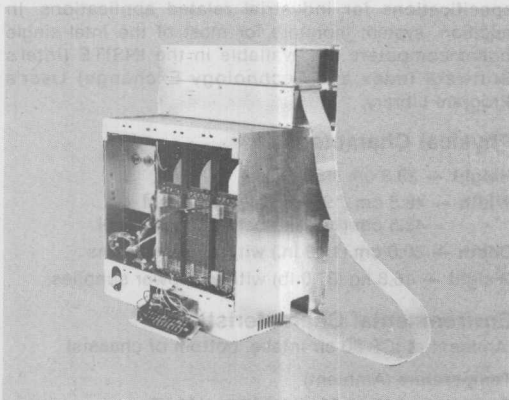


Figure 4. Rear View iCS 80 Chassis Showing Power Distribution Panel, and Cabling from iCS 80 Chassis to iCS 9XX RETMA Mounted Signal Conditioning Panels (Top of iCS 80 Chassis)

Typical Small Configuration

- 8-bit 8088 processor (iSBC 88/40)
- 4K bytes RAM (8K optional)
- 2K-16K bytes E²PROM or 8K-128K bytes ROM/EPROM
- 16-64 analog inputs
- 8 analog outputs
- 8 isolated digital inputs
- 8 isolated digital outputs

OR

- 12 TTL outputs
- 12 TTL inputs

Typical Maximum Configuration

- 16-bit 8086 processor (iSBC 86/30 w/RAM MULTIMODULE)
- 768K bytes RAM (2 - iSBC 056)
- 128K bytes EPROM (or 16K E²PROM)
- 240 analog inputs (3 - iSBC 88/40 w/2 ea. iSBC 311)
- 24 analog voltage outputs

OR

- 24 analog current outputs (4-20 mA)
- 72 isolated digital inputs/outputs
- 144 TTL digital inputs/outputs (2 - iSBC 519s)

(All iCS 9XX Signal Conditioning/Termination Panels are not shown)

Lockable Service Panel

To assist in development, checkout and service, two pushbuttons are provided. The RESET button pulls low the initialize line (INIT) on the MULTIBUS backplane. The INTERRUPT button pulls low one interrupt line on the MULTIBUS backplane (INT1). Logic within the iCS 80 ensures that these buttons function with all versions of Intel single board computers. From the front of the iCS 80 chassis, without a CRT or other panel, an operator or service person can reset or interrupt on-going iCS 80 system operations to get attention, signal an alarm, or start a self-test operation.

A front panel key provides three positions: OFF (AC power off and key removable), ON (AC power on, pushbuttons enabled, key unremovable), and LOCK (AC power on, pushbuttons disabled, key removable).

Three indicator light emitting diodes record basic chassis status. POWER ON (GREEN); RUN (GREEN); and HALT (RED); the RESET or INTERRUPT buttons will remove the HALT state.

U.L. Approved

The iCS 80 chassis has received full Underwriters Laboratory approval (F.6 #E70842) as a U.L. listed component under the Underwriters Laboratories Safety Standard for

Process Control Equipment, UL1092. When installed as described in the iCS 80 Installation Manual, the iCS 80 chassis provides adequate protection against shock, fire and casualty hazards, and should comply with most local and regional requirements for installation in ordinary locations. In addition, the iCS 80 chassis was designed to comply with the UL requirements for Data Processing Equipment, UL478. It has also been submitted to the Canadian Standards Association and approval is pending under CSA category C22.2 No. 142, the Canadian Standard for Safety for Process Control Equipment and C22.2 No. 154 for Data Processing Equipment.

Mounting Space for Signal Conditioning/Wire Terminations

The cardcages and power supplies in the iCS 80 chassis are recessed behind the front edge of the rack mounting ears to provide mounting space for the iCS 9XX series signal conditioning/termination panels and field wiring. For smaller systems with only one or two iSBC 604/614 cardcages (4 to 8 slots), up to two iCS 910, iCS 920, or iCS 930 signal conditioning/termination panels can be mounted vertically over the area where the second or third cardcage would mount (see Figure 2). The benefit of this design is a completely self-contained industrial chassis with iSBC cards, power supply, signal conditioning and field wiring terminations, all in one enclosure.

SPECIFICATIONS

Capacity

Four slots for MULTIBUS compatible single board computers, memory, I/O or other expansion boards
Expandable to 12 slots using customer plug-together iSBC 614 cardcages

Front Panel Controls

Pushbuttons

RESET: Connected to Initialize/ on MULTIBUS backplane

INTERRUPT: Connected to Interrupt 1/ line on MULTIBUS backplane.

Panel Indicator Lights (LEDs)

POWER ON (green): +5V power exists on the MULTIBUS backplane

RUN (green): CPU is executing an instruction. Light goes out if CPU is in WAIT or HALT state

HALT (red): CPU has executed a HALT instruction

Keylock

OFF: AC power off, key removable

ON: AC power on, pushbuttons enabled, key unremovable

LOCK: AC power on, pushbuttons disabled, key removable

Fuse — AC power (6A)

Equipment Supplied

iCS 80 industrial chassis, three fans for cardcages, one fan for power supply, 4-slot cardcage with MULTIBUS backplane, control panel with switches, indicators, keylock, power distribution barrier strip, AC power fuse, line filter, 115V power cable, and logic for interrupt and reset buttons. An installation package is also provided, including a NEMA cabinet mounting kit, power supply extension cables, and RETMA cabinet mounting screws, 110/230 VAC operation.

Software

See the RMX/80 Real-time Multitasking Executive specifications for industrial related applications. In addition, system monitors for most of the Intel single board computers are available in the INSITE (Intel's Software Index and Technology Exchange) User's Program Library.

Physical Characteristics

Height — 39.3 cm (15.7 in.)

Width — 48.5 cm (19.0 in.) at front panel
43.5 cm (17.4 in.) behind front panel

Depth — 30.0 cm (12.0 in.) with all protrusions

Weight — 16.8 kg (37.0 lb) without power supplies

Environmental Characteristics

(Ambient at iCS-80 air intake, bottom of chassis)

Temperature (Ambient)

Operating: 0°C to 50°C (32°F to 122°F)

Non-operating: -40°C to +85°C

Humidity — Up to 90% relative, noncondensing at 40°C

ICS 80

Electrical Characteristics

The ICS 80 chassis provides mounting space for either the iSBC 635 or iSBC 640 power supply. Unless otherwise stated, electrical specifications apply to both power supplies when installed by user in ICS 80 chassis.

Input Power

Frequency: 47 to 63 Hz. Voltage (Nominal)

(Single Phase): 100, 115, 215, or 230 VAC +10%, jumper selectable.

Current: (Including fans)	With iSBC 635	With iSBC 640	Input Voltage
	3.0A max	5.6A max	
	1.5A max	2.8A max	206 VAC
Power, max:	315 watts	580 watts	

Output Power

Voltage	Output Current (max)		Overvoltage Protection	
	iSBC 635	iSBC 640	iSBC 635	iSBC 640
+12V	2.0A	4.5A	+14V to +16V	+14V to +16V
+5V	14.0A	30.0A	+5.8V to +6.6V	+5.8V to +6.6V
-5V	0.9A	1.75A	-5.8V to -6.6V	-5.8V to -6.6V
-12V	0.8A	1.75A	-14V to -16V	-14V to -16V

Combined Line/Load Regulation — $\pm 1\%$ at $\pm 10\%$ static line change and $\pm 50\%$ static load change, measured at the output connector ($\pm 0.2\%$ measured at the power supply under the same conditions).

Remote Sensing — Provided for +5 VDC output line regulation.

Output Ripple and Noise — 10 mV (iSBC 635 and iSBC 640

supply) peak-to-peak, max (DC to 500 kHz)

Output Transient Response — Less than 50 μsec for $\pm 50\%$ load change.

Maximum Watts Dissipation (load plus losses) — 500W (iSBC 640 supply), 250W (iSBC 635 supply)

Installation

Complete instructions for installation are contained in the ICS 80 Site Planning and Installation Guide, including RETMA and NEMA cabinet mounting, and field signal, ground wiring and cooling suggestions.

Warranty

The ICS 80 Industrial Chassis is warranted to be free from defects in materials and workmanship under normal use and service for a period of 90 days from date of shipment.

Reference Manuals

9800799A — ICS 80 Industrial Chassis Hardware Reference Manual (SUPPLIED)

9800798 — ICS 80 Industrial Systems Site Planning and Installation Guide (SUPPLIED)

9800708A — iSBC 604/614 Cardcage Hardware Reference Manual (SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

The ICS 80 Industrial Chassis must be ordered as a kit with an Intel power supply of your choice. Ordering as a kit will ensure shipment of the power supply and ICS 80 chassis at the same time. Typical configurations and ordering instructions are:

Part Number Description

ICS 80 Kit 635	iCS 80 system consisting of: iCS 80 Industrial Chassis iSBC 635 Power Supply
ICS 80, Kit 640	iCS 80 system consisting of: iCS 80 Industrial Chassis iSBC 640 Power Supply



APPLICATION NOTE

AP-52

Maximum Watts Dissipation (load plus losses) — 500W
(58C 640 supply), 250W (58C 635 supply)
Output Transient Response — Less than 50 μ sec for 100%
load change.

Installation

Complete instructions for installation are contained in
the ICS 80 Site Planning and Installation Guide, in-
cluding RETMA and NEMA cabinet mounting and field
signal, ground wiring and cooling suggestions.

Warranty

The ICS 80 Industrial Chassis is warranted to be free
from defects in materials and workmanship under nor-
mal use and service for a period of 90 days from date of
shipment.

Reference Manuals

9800700A — ICS 80 Industrial Chassis Hardware Refer-
ence Manual (SUPPLIED)
9800705B — ICS 80 Industrial Systems Site Planning and
Installation Guide (SUPPLIED)
9800705A — 58C 6040/58C 6045 Cardcage Hardware Refer-
ence Manual (SUPPLIED)
Reference manuals are shipped with each product only.
If designated SUPPLIED (as above), Manuals may be
ordered from any Intel sales representative, distributor
office or from Intel Literature Department, 3065 Bowers
Avenue, Santa Clara, California 95051.

The ICS 80 chassis provides mounting space for either the
58C 635 or 58C 640 power supply. Unless otherwise
stated, electrical specifications apply to both power
supplies when installed by user in ICS 80 chassis.

Input Power

Frequency: 47 to 63 Hz Voltage (Nominal):
(Single Phase): 100, 115, 230 or 250 VAC $\pm 10\%$ (single
phase)

Current (Maximum)	With 58C 635 Supply	With 58C 640 Supply	Input Voltage
3.0A max.	3.0A max.	2.5A max.	100 VAC
1.5A max.	1.5A max.	2.5A max.	230 VAC
315 watts	315 watts	350 watts	

Output Power

March 1979

Voltage	58C 635	58C 640	58C 635	58C 640
+15V	3.0A	2.5A	+15V to +15V	+15V to +15V
+5V	1.5A	1.5A	+5V to +5V	+5V to +5V
-5V	0.8A	1.2A	-5V to -5V	-5V to -5V
-15V	0.8A	1.2A	-15V to -15V	-15V to -15V

Combined Line Load Regulation — $\pm 1\%$ at 10% static
load change and 50% static load change, measured at the
output connector (1.0 μ sec measured at the power supply
under the same conditions).

Remote Sensing — Provides 15 VDC output line
regulation.

Output Ripple and Noise — 10 mV rms and 58C 640

Using Intel's Industrial Control
Series in Control Applications

Peter Andersen
OEM Microcomputer Systems Applications

I. INTRODUCTION

The introduction of the single board computer as a tool for the system designer has opened the way for many varied application areas to benefit from the advantages of computer utilization. A problem still exists, however, because the available I/O configurations have been largely incompatible with the wiring and packaging techniques required in industrial environments. This problem is overcome by the utilization of the Intel® iCST™ product family. The purpose of this application note is to provide a representative approach to the implementation of a computerized solution to an industrial control system.

System Description

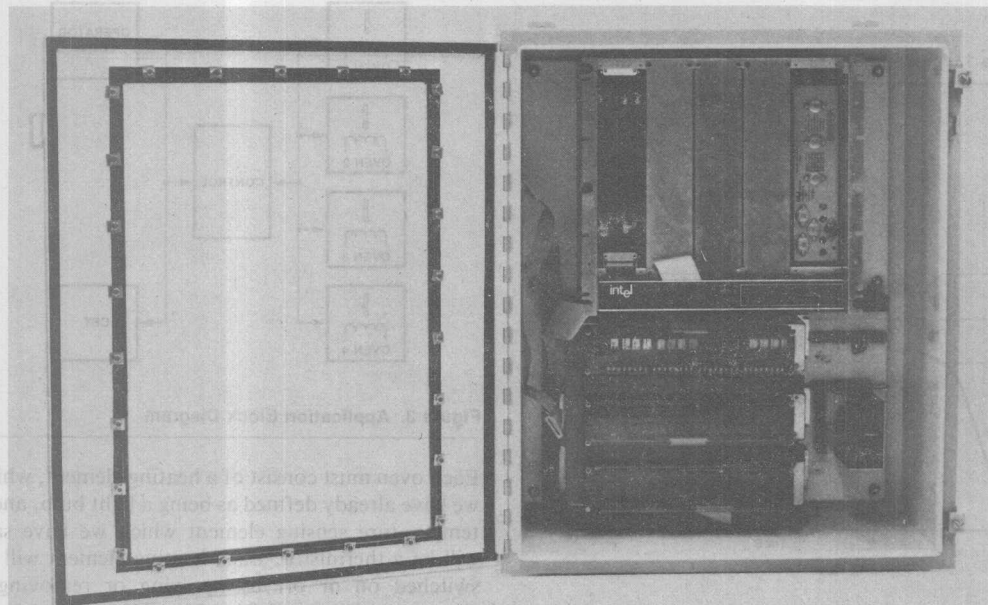
This application note will deal with a control system which will regulate the temperature in each of four ovens. Each oven will be defined as utilizing a light bulb for heating. Normal convection will be used to provide cooling. The internal temperature will be measured by means of a thermistor installed in each oven. We will assume that we will be required to implement some type of operator panel near the ovens which will allow the status of each oven to be monitored. This approach is similar to many common industrial applications which require a supervisory control station in one area and a separate operator interaction panel near the

equipment being controlled. The setpoint and tolerances should be input from an external location.

With these facts about our system defined, we can begin a step by step solution to providing a computerized control system to operate the ovens. We will discuss the various equipment trade-offs and the decisions which will be used to define the hardware/software designs.

Control Algorithm

Before we can begin the design of our system, we must have a clear idea of the technique we will use to control the system. Our control system must maintain the oven temperature within a predefined and fairly narrow range of the setpoint. Let us make an assumption that the light bulb will be controlled digitally, meaning that the bulb must either be turned fully on or it must be turned fully off. The obvious control technique then becomes turning the bulb on when the temperature of the oven is below our lower limit and turning the bulb off when the temperature is above the higher limit. It seems reasonable to assume that this technique will provide a temperature in the oven which varies sinusoidally with time. This is true because even though the lamp is turned off, it will continue to generate heat for a short period of time. Likewise, when the bulb is turned on, it will not instantly be able to provide heat to raise the temperature of the



chamber. We would expect to have a system response such as is shown in Figure 1. A better method of control can be devised if we provide some type of temperature prediction into our control algorithm. Since this utilizes the rate of temperature increase or decrease, it will involve a type of derivative control system. This derivative control action will tend to dampen the temperature oscillations which might be encountered if only an instantaneous on-off control system were utilized. Figure 2 shows the response with time that we might expect with this type of control system.

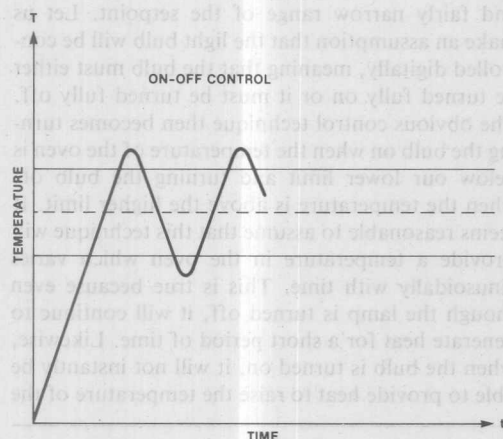


Figure 1. Maximum Effort Current Temperature

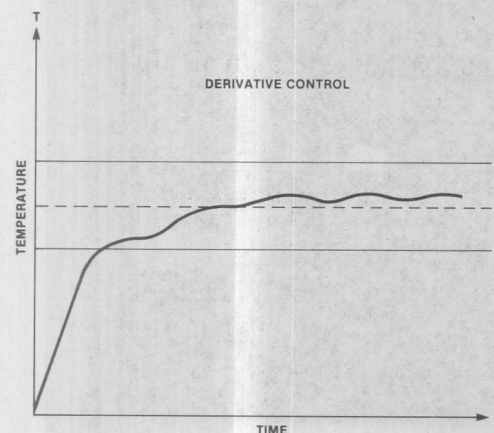


Figure 2. Maximum Effort Projected Temperature

The second approach is superior to the first because the control will provide a much smaller oscillation of the oven temperature. Other solutions are possible, such as providing a modulated output to the lamp. However, in an attempt to provide a simple model upon which to expand our system solution, we will assume that the second approach will provide us with an accurate enough control of the oven temperature.

Having made the decision as to the control technique, we can proceed with the task of determining the general system configuration. That is, we can define the physical system characteristics and the components to which we must interface the computer system. This approach is identical to that which would be used in a conventional control system design.

Basic System Configuration

Based upon the data which we have provided so far, it is possible to build a block diagram of the system's major components. The system consists of four ovens, an operator's panel, a data entry panel, and the actual control logic. A block diagram for the system is shown in Figure 3. We must now further define the elements which make up each of these blocks.

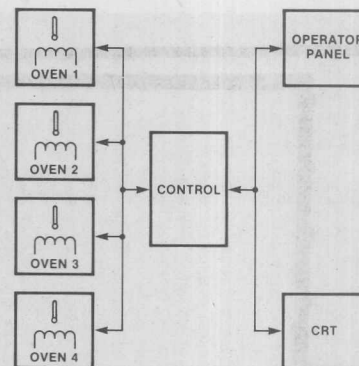


Figure 3. Application Block Diagram

Each oven must consist of a heating element, which we have already defined as being a light bulb, and a temperature sensing element which we have said will be a thermistor. Each heating element will be switched on or off by applying or removing a source of 115 VAC. The thermistor temperature can be sensed by using the thermistor in a voltage

divider circuit. We can then measure the voltage across a fixed resistor to obtain an analog signal which is proportional to the oven temperature. We will determine the required value of the fixed resistor at a later time.

The operator's panel should be designed to provide the workfloor operator with basic information as to the status of each oven. It should also allow some method by which he can inhibit the operation of any oven should it become necessary for charging or servicing the oven. We can then define the basic elements which should make up the operator's control panel. Each oven should have associated with it the following controls and indicators:

1. Oven ON/OFF Switch — This switch will allow the operator to inhibit the oven operation by turning the appropriate oven switch to OFF.
2. Oven RUNNING Indicator — This indicator will provide a visual indication that the oven is activated and that the temperature is being controlled.
3. Oven IN TOLERANCE Indicator — This indicator will turn on when the oven temperature falls within the allowable bandwidth around the setpoint for that oven.
4. Oven ALARM Indicator — This indicator is the complement of the in tolerance lamp. It will be turned on when the oven is activated and the temperature does not lie within the desired bandwidth.
5. Oven CAUTION Indicator — It may be necessary to alert the operator to a potential oven temperature control problem before it actually occurs and sets off the alarm indicator. Since we have defined our control algorithm as utilizing a type of derivative control, we can project the oven temperature ahead in time. We will turn the oven caution indicator on when we predict that the oven temperature will lie outside of the desired bandwidth in a predetermined future time period.

We have now defined the operator interface which we will utilize to control and monitor the oven processes.

At this point, we will make a decision that the interface used to input the setpoints will utilize a CRT terminal. Though the decision may seem to be completely arbitrary, we will see later that CRT terminals provide an extremely useful device for allowing an operator to communicate with the system. Once the decision has been made, we have no

further requirements to consider hardware design for this terminal, as the entire operation can be handled in the software development which will be considered later.

A common technique for documenting a system is the ladder diagram. At this time, we can construct a ladder for our control system. Unlike conventional design techniques, our ladder diagram need only be concerned with the actual drive and sensing circuits since the logic required to drive the various outputs will be defined using software. This results in a considerable simplification of the design process. A ladder diagram for a typical oven is shown in Figure 4. We can defer the implementation of the control algorithm until we begin to develop the software portion of our control system. It is now possible to complete the external hardware design and to implement the system wiring package.

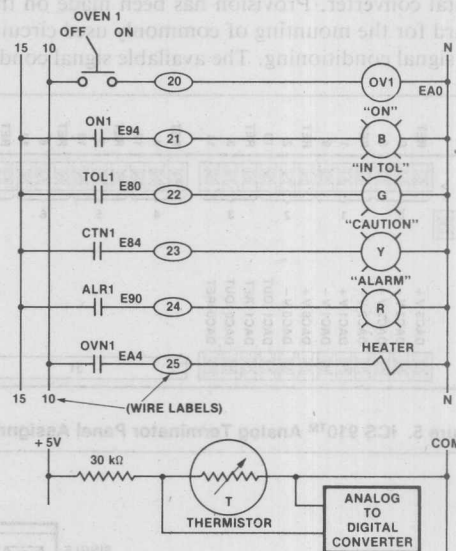


Figure 4. Ladder Diagram of One Oven

II. WIRING INTERFACES

A major pitfall in utilizing a computer for control systems has traditionally been the requirement for the design engineer to expend a considerable amount of his time in designing interfaces to connect the physical wiring to the computer system. The introduction of Intel's product line of termination panels has essentially eliminated the require-

ment of designing interfaces and allows more engineering time to be spent providing a solution to the application. Before we continue with the specific design, we should spend some time discussing the various types of termination panels available and the general characteristics of each panel.

Analog Termination Panels

The Intel® iCS 910™ Analog Termination Panel has been designed to provide a simple means of terminating the analog wiring and of providing an interface to the control system input/output. All wiring is terminated utilizing pressure type screw barrier blocks. Termination blocks have been provided to allow the termination of up to 32 single-ended or 16 differential channels of analog input. For use in a differential input environment, such as we will be using, the terminator blocks provide wiring terminations compatible with shielded cable inputs in that provision has been made to accept the shield of each input signal. The shield is then carried through the on-board circuits to the analog-to-digital converter. Provision has been made on the board for the mounting of commonly used circuits for signal conditioning. The available signal condi-

tioning circuits provide for installation of current termination resistors and the installation of a single pole low pass filter network. The basic barrier assignments for the iCS 910 termination panel are shown in Figure 5. The possible circuit networks for this panel are illustrated in Figure 6. A complete description of the analog termination panel can be found in the *iCS 910 Analog Signal Conditioning/Termination Panel Hardware Reference Manual* (manual order number 9800800A).

The functions of the analog termination panel will become more clear as we develop the actual configuration required to support our oven application. Referring to the ladder diagram (Figure 4) we see that a fixed resistor is necessary to provide the voltage divider network to sense the oven temperature. The current termination resistor (R_c) on the iCS 910 board can be used to provide a convenient mounting location for this component (refer to iCS 910 circuit schematic, Figure 6). At this point, we must make a design decision regarding the utilization of a low pass filter for our analog circuits. Since the oven temperatures are not expected to exhibit rapid fluctuations with time, the use of a low pass filter will not adversely effect the temperature

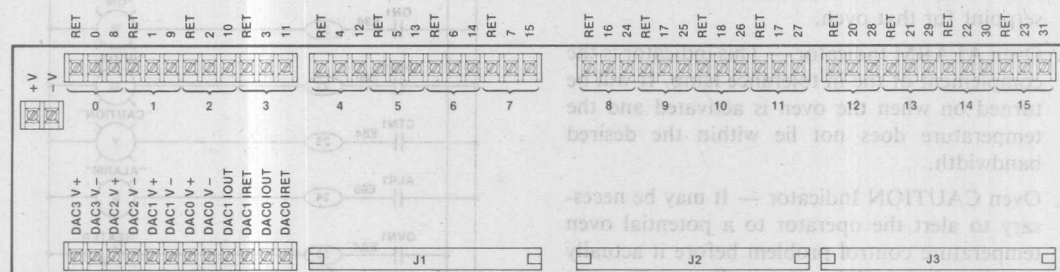


Figure 5. iCS 910™ Analog Terminator Panel Assignments

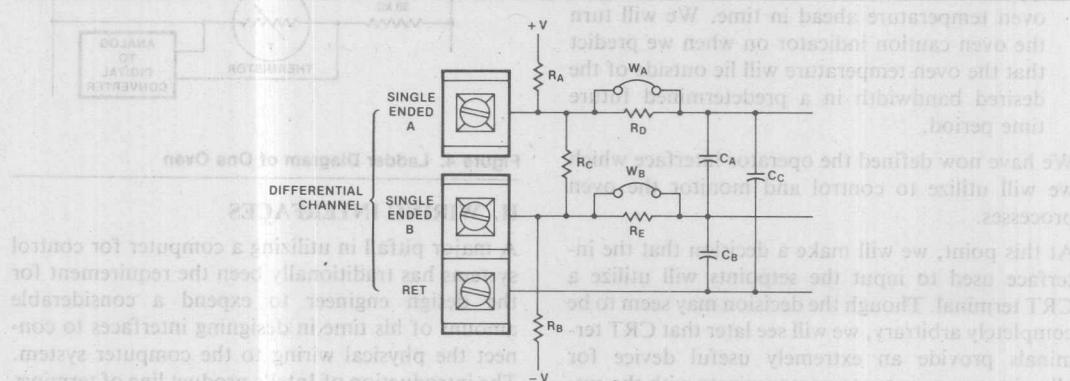


Figure 6. Typical Circuit on Analog Terminator

sensing. Indeed, the use of a low pass filter should contribute to spurious signal rejection should the analog cables pick up external noise signals. Calculations will show that the use of a filter network consisting of 11K ohm series resistors and a 2.2 μ F capacitor will provide the filter characteristics shown in Figure 7.

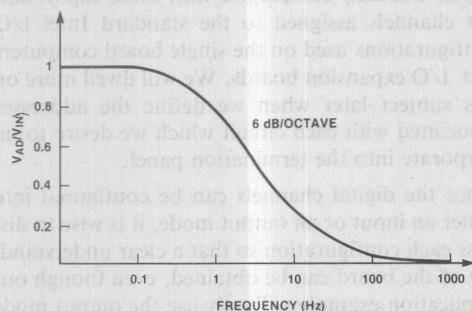


Figure 7. Single Pole Filter Characteristics

Based upon our requirements and using the circuit schematic of Figure 6, we can provide the circuit interfaces required by our ladder diagram (Figure 4) by configuring the channels of the iCS 910 terminator as shown in Figure 8. This results in a simple two-wire per oven analog interface. The terminator board is designed to connect to the various analog I/O boards by means of a standard ribbon

cable which is supplied with the terminator panel. The actual selection of the appropriate analog board will be deferred until later. We will define that oven number 1 will correspond to the differential analog channel 0; oven 2 will correspond with channel 1; oven 3 will correspond with channel 2; and oven 4 will use channel 3. This leaves 12 analog differential channels available for future expansion. The channel selection just made was a purely arbitrary choice.

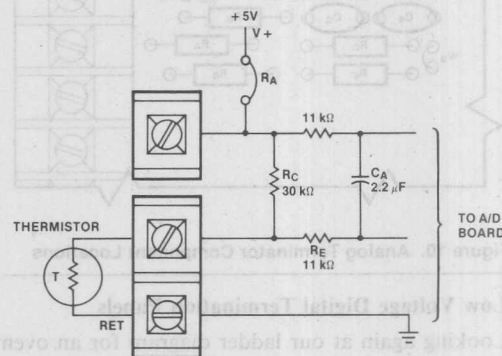


Figure 8. Analog Circuit for Oven Application

The wiring to the iCS 910 terminator panel can then be made essentially as shown in Figure 9. Clearly, the use of the terminator panel greatly simplified the connection between the control sys-

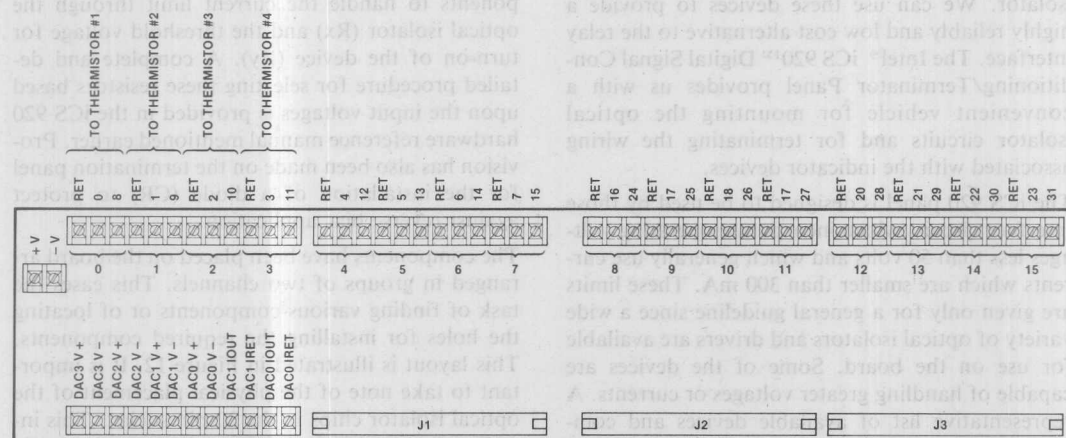


Figure 9. Analog Terminator Wiring

tem and the physical devices which are to be monitored or controlled. Figure 10 shows the placement of the components onto the board.

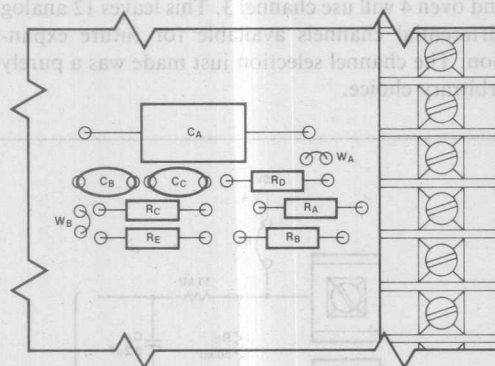


Figure 10. Analog Terminator Component Locations

Low Voltage Digital Termination Panels

Looking again at our ladder diagram for an oven control system (Figure 4), we see the need to provide a second type of interface signal. This is to provide the switching for the various indicator lamps used on the operator's control panel. Traditionally, this interface has been handled by using electromechanical relays. The coils would be driven by the low voltage control system and the relay contacts were used to drive the external indicators. Modern technology provides us with a solid state device to perform the same function, the optical isolator. We can use these devices to provide a highly reliably and low cost alternative to the relay interface. The Intel® iCS 920™ Digital Signal Conditioning/Terminator Panel provides us with a convenient vehicle for mounting the optical isolator circuits and for terminating the wiring associated with the indicator devices.

The iCS 920 panel is designed to be used by those interface circuits which incorporate operating voltages less than 50 volts and which generally use currents which are smaller than 300 mA. These limits are given only for a general guideline since a wide variety of optical isolators and drivers are available for use on the board. Some of the devices are capable of handling greater voltages or currents. A representative list of available devices and complete details of the termination panel are available in the *iCS 920 Digital Signal Conditioning/Termination Panel Hardware Reference Manual* (manual order number 9800801A).

The digital panel provides terminations for up to 24 digital channels, each of which can be configured as either an input or an output channel according to the specific application requirements. As with the analog termination panel, all wire terminations are made using pressure type barrier strips which will accept up to 16 gauge wire. The 24 digital channels correspond with those input/output channels assigned to the standard Intel I/O configurations used on the single board computers and I/O expansion boards. We will dwell more on this subject later when we define the addresses associated with each circuit which we desire to incorporate into the termination panel.

Since the digital channels can be configured into either an input or an output mode, it is wise to discuss each configuration so that a clear understanding of the board can be obtained, even though our application example will only use the output mode with this board.

Figure 11 provides a schematic of the panel when it is configured for a digital input mode. To set up a channel to operate as an input, it is necessary to add at least two jumpers to the wire-wrap jumper posts. As can be seen, pins 6 and 4 must be connected together as well as pins 3 and 5. If the board is to provide a visual LED indication of the channel status, an additional jumper should be installed between pins 1 and 2 of the jumper posts. If this is done, be certain to take into account the additional current requirements when calculating the required input resistors. Two resistor mounting locations are provided to allow installation of selected components to handle the current limit through the optical isolator (Rx) and the threshold voltage for turn-on of the device (Ry). A complete and detailed procedure for selecting these resistors based upon the input voltages is provided in the iCS 920 hardware reference manual mentioned earlier. Provision has also been made on the termination panel for the installation of a diode (CR) to protect against reverse bias application.

The components have been placed on the board arranged in groups of two channels. This eases the task of finding various components or of locating the holes for installing the required components. This layout is illustrated in Figure 12. It is important to take note of the physical placement of the optical isolator chips in the 20-pin socket. This installation location must be followed rigorously when using a channel in an input mode. Also take note that provisions are provided for mounting two sizes of resistors in location (Rx). This will accom-

modate the power dissipation requirements which will be encountered in various application situations. Referring again to Figure 12, note that the upper half of the layout represents odd channels and the lower portion of the layout is used for even channel component mounting.

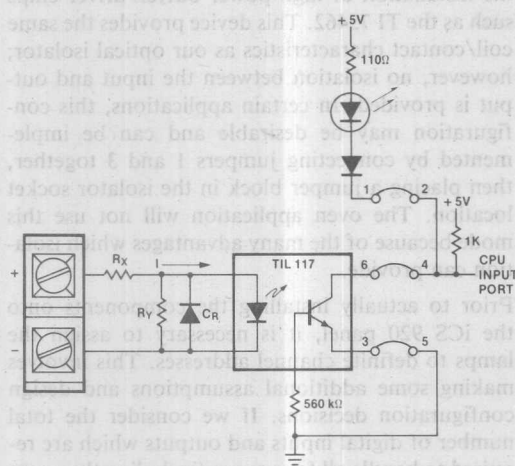


Figure 11. iCS 920™ Digital Terminator Input Configuration

When the iCS 920 panel is used in this input mode, it corresponds to the utilization of a relay coil to sense some external contact closure. The resistors can be thought of as selecting the coil's operating voltage and the diode provides the same transient

protection function as when installed on an electro-mechanical relay. Finally, the optical isolator output corresponds to the contacts associated with the relay coil. As we will see later, this approach provides us with an unlimited number of contacts per relay coil.

The oven application requires a contact for driving the indicator lamps associated with each oven. If we define the driving voltage to be 24 volts DC, we will find that the voltage and current requirements fall within the limits specified for using the iCS 920 Digital Signal Conditioning/Termination Panel. Let us examine in more detail how this can be accomplished.

We will select an industrial indicator assembly which utilizes a full voltage 24-volt lamp. Typical lamps would be type 387. This will require a drive of 40 mA at 28 volts. Our switching device must be capable of driving this load. The analogy used earlier to compare the optical isolator with a relay in an input mode holds true when we utilize the devices in an output configuration. If we examine the data sheet for the current switching characteristics of a typical optical isolator, say the TIL 113 (Appendix A), we can see that the current and voltage requirements fall well within the allowable ratings of the device. We have selected the relay contact characteristics! We need not concern ourselves with the selection of current limitation resistors (coil voltage ratings) since this circuitry is provided on the terminator panel when a circuit is

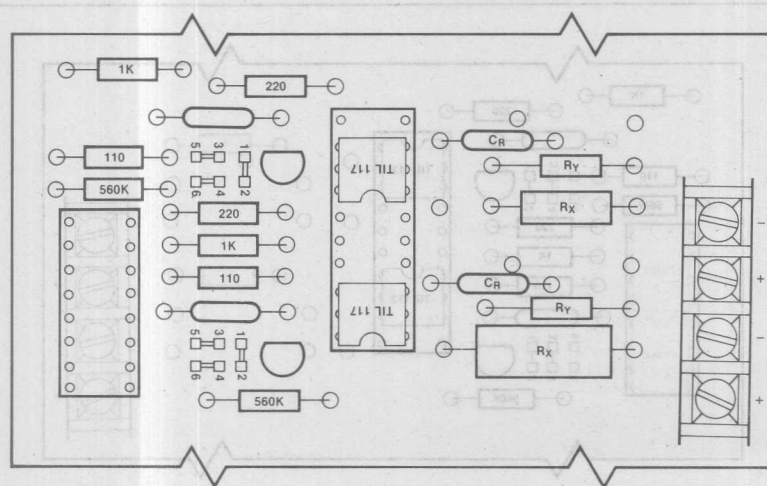


Figure 12. Digital Terminator Input Parts Layout

configured in an output mode. If we refer to Figure 13, we can see the on-board schematic for the output drive mode of operation. Two jumpers must be installed for each output channel. The first, between pins 1 and 2, is used to enable the LED channel status indicator. The second, between pins 3 and 4, actually connects the computer generated drive signal to the input of the optical isolator (analogous to connecting the relay coil to the driving line). Provision has been made on the circuit board for only one optional component in the output mode; this is the resistor (R_z). This component has the effect of increasing the response time of the switching device. Because our indicator lamps are not time critical, we will choose to omit the installation of this component.

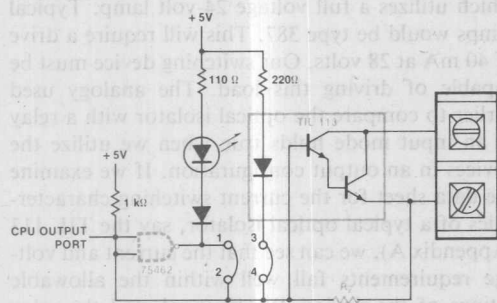


Figure 13. iCS 920™ Digital Terminator Output Circuit

Figure 14 provides a drawing showing the location of the components on the iCS 920 panel when it is utilized as an output switch. Again note the place-

ment of the optical isolators in the 20-pin sockets. Also note the jumper arrangement used to provide the required output circuitry.

Again referring to Figure 13, we see that an alternative to using the optical isolator for a switch exists. Provision has been made on the panel for the installation of high power buffer/driver chips such as the TI 75462. This device provides the same coil/contact characteristics as our optical isolator; however, no isolation between the input and output is provided. In certain applications, this configuration may be desirable and can be implemented by connecting jumpers 1 and 3 together, then placing a jumper block in the isolator socket location. The oven application will not use this mode because of the many advantages which isolation can provide.

Prior to actually installing the components onto the iCS 920 panel, it is necessary to assign the lamps to definite channel addresses. This involves making some additional assumptions and design configuration decisions. If we consider the total number of digital inputs and outputs which are required to handle all four ovens (including the as yet unconsidered switch and heater signals), we see that a total of 24 channels will be required. These will be broken out as shown below:

No. of Channels	Type	Function
16	DC	Oven indicator lamps
4	AC	Oven heaters
4	AC	Oven RUN switches

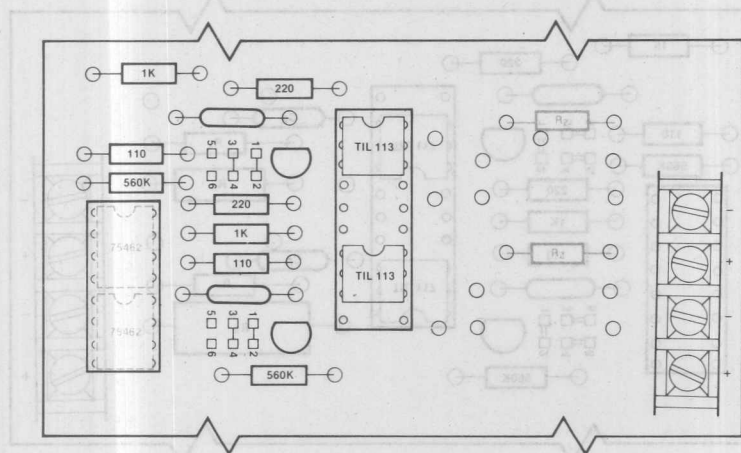


Figure 14. Digital Terminator Output Configuration

We have indicated that the 16 indicator lamps can be handled using the iCS 920 panel. An examination of the data sheets for the various Intel single board computers and expansion boards provides us with the fact that a common characteristic of most boards is the use of at least one Intel 8255 Programmable Peripheral Interface. This provides us with at least 24 I/O lines with which to work on each single board computer. We can then assume that we will not require an I/O expansion board to implement our application. Ideally, we can handle our total requirements with one parallel interface.

The various Intel parallel ports are brought off of the computer and expansion boards using edge connectors. These edge connectors are then connected to the termination panels using a standard ribbon cable assembly, effectively providing an extension of the I/O ports out to the termination panels. The 24 channels are grouped into three I/O ports (each consisting of 8 channels or bits) which are then called port A, port B, and port C. When connected to the iCS 920 panel, these ports and their bit assignments will be as shown in Figure 15.

At this point, we seem to be in a dilemma since we would like to use all 24 channels and we have used only 16 of them on our panel while we have utilized the edge connector of the interface. It would be desirable to have some technique to extend the other 8 channels to a high voltage terminator panel. It might be well to interrupt our channel assignments at this time to jump ahead and consider the features of the iCS product line which will enable us to accomplish our interface desires. We will then consider the interface of the high voltage signals to our control system before returning to the problem of assigning port locations to our lines.

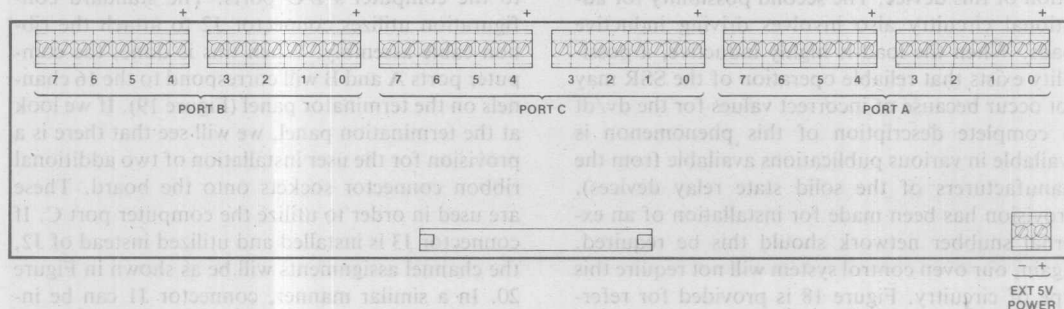


Figure 15. iCS 920™ Digital Terminator Port Assignment

High Voltage Digital Termination Panels

The Intel® iCS 930™ AC Signal Conditioning/Termination Panel is designed to interface up to 16 AC signals (up to 280 volts at 3 amps) or high current DC signals (up to 50 volts at 3 amps) to the parallel ports of the Intel single board computers or I/O expansion modules. The barrier strip terminations on this panel are designed to easily handle the 14 gauge wire commonly found in applications requiring the use of the AC terminator.

Solid state relays are used to provide the interface between the computer I/O ports and the physical plant devices. These devices make the utilization of the panel a simple task once a ladder diagram of the required circuits has been drawn. As we have previously mentioned and as is clear from looking at Figure 4, we shall need to utilize eight of the available circuits, four for input and four for output. The implementation of each signal type requires only that we insert the correct type of solid state relay into the appropriate socket.

First, consider the input configuration which is required to sense the position of the oven RUN switches. Figure 16 shows the circuit schematic when used in the input mode. We can see that the output signal will turn on when the input power is applied. Like the digital termination panel, each circuit's status is indicated by means of an LED indicator installed on the board. The input circuit is protected by a socketed 3-amp fuse which may be replaced without the need to solder any components. The solid state relay used for this configuration should be a type IAC5 which is available from either Opto-22 or Motorola. Complete details of available relays and their uses on the board are available in the *iCS 930 AC Signal Conditioning/*

Termination Panel Hardware Reference Manual (manual order number 9800802A). Keep in mind the fact that although this application note represents the solid state relays as being actual relays and contacts, they in fact are solid state and contain no moving parts.

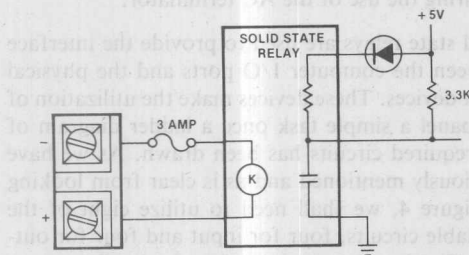


Figure 16. iCS 930™ AC Terminator Input Circuit

The output configuration is utilized to turn the heater elements (the light bulbs) on and off. Figure 17 provides us with a schematic of the output circuitry. In this case, we will insert a solid state relay of type OAC5 which will handle up to 140 volts RMS at 3 amps. In some cases, it might be desirable to add certain components to the terminator panel when using it in the output mode. Two possible circuit configurations are possible. The first and perhaps the most common will consist of installing a MOV (metal oxide varistor) across the solid state relay contacts. This will be required when the load being driven is inductive in order to prevent the transients generated by the load from damaging the triac in the SSR (solid state relay). Since the SSRs utilize zero voltage switching and the load in our ovens is resistive rather than inductive, our application will not necessitate the installation of this device. The second possibility for additional circuitry also involves driving inductive loads. When the load is highly inductive, a possibility exists that reliable operation of the SSR may not occur because of incorrect values for the dv/dt (a complete description of this phenomenon is available in various publications available from the manufacturers of the solid state relay devices). Provision has been made for installation of an external snubber network should this be required. Again, our oven control system will not require this type of circuitry. Figure 18 is provided for reference should the reader desire to see the location of the additional components on the panel. It should be noted that the component placement does not

allow the installation of the MOV and the snubber simultaneously.

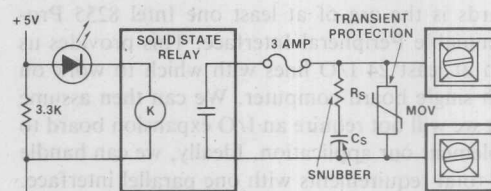


Figure 17. iCS 930™ AC Terminator Output Circuit

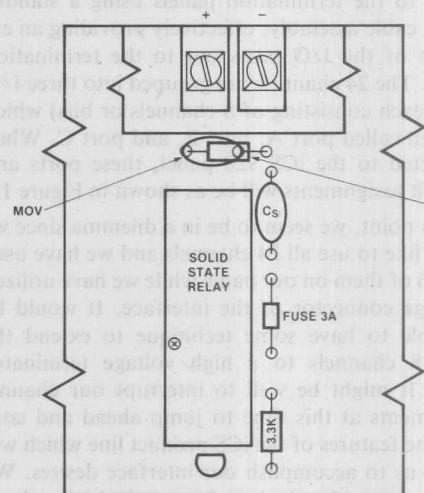


Figure 18. AC Terminator Component Locations

We can now get back to the task of assigning addresses to the various digital channels. The iCS 930 panel has three connector options for connecting it to the computer's I/O ports. The standard configuration utilizes connector J2 to attach the ribbon cable assembly. When this is done, the computer ports A and B will correspond to the 16 channels on the terminator panel (Figure 19). If we look at the termination panel, we will see that there is a provision for the user installation of two additional ribbon connector sockets onto the board. These are used in order to utilize the computer port C. If connector J3 is installed and utilized instead of J2, the channel assignments will be as shown in Figure 20. In a similar manner, connector J1 can be installed and utilized to provide connections between the computer port C and the other eight SSR positions. If we choose the 16 lines required for driving

the indicator lamps from the iCS 920 panel to be ports A and B, then it seems reasonable to assign the eight remaining lines required on the iCS 930 to port C. A feature of utilizing standard ribbon cable assemblies is the ability to easily add ribbon plug connectors to the cable. This will result in an assembly transferring ports A, B and C to the iCS 920 panel (however, port C is not used) and which continues the port C signals to the iCS 930 panel.

Individual channel assignments can now be made, grouping the inputs and outputs together in groups of four (this is done because of a requirement of the single board computers to share terminator and driver component packages in groups of four). Figure 21 provides a drawing showing the channel assignments and the physical wiring locations which will be used to connect the oven heaters and switches.

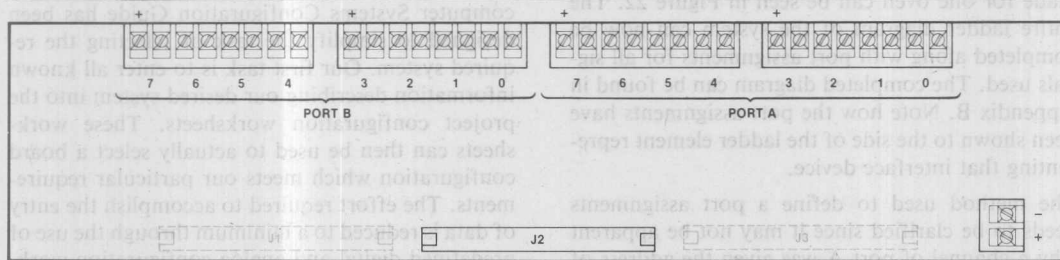


Figure 19. iCS 930™ AC Terminator Port Assignments

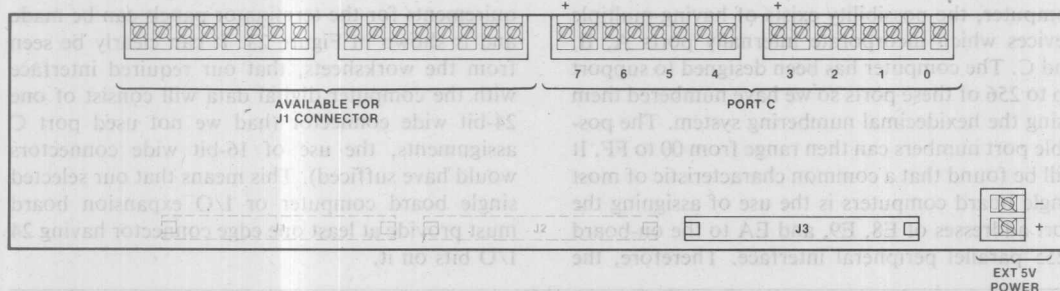


Figure 20. iCS 930™ AC Terminator Port Assignments

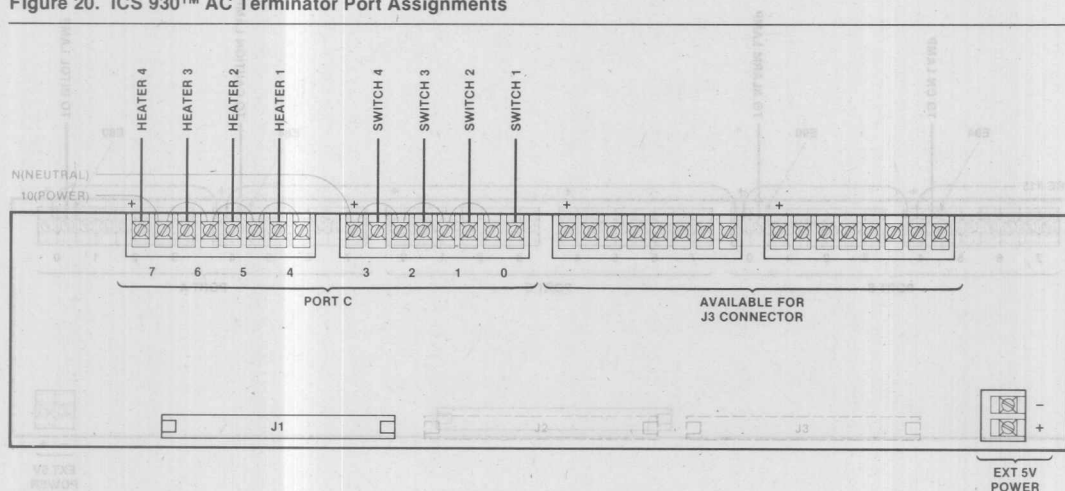


Figure 21. iCS 930™ AC Terminator Application Configuration

Final Channel Assignments

The only task remaining before we have completed our task of assigning channel numbers and physical wire and component locations is to assign these channels on the iCS 920 digital termination panel. Since we have already determined that we will utilize ports A and B, this becomes a simple matter, requiring only an arbitrary assignment of lamp locations using these port bits. The assignments made for one oven can be seen in Figure 22. The entire ladder diagram of the system can now be completed along with port assignments for all signals used. The completed diagram can be found in Appendix B. Note how the port assignments have been shown to the side of the ladder element representing that interface device.

The method used to define a port assignments needs to be clarified since it may not be apparent why a channel of port A was given the address of E80. To begin, we have already indicated that each port consisted of eight channels or bits. We will number these bits from 0 to 7. Since it is possible to have many input/output devices connected to the computer, the possibility exists of having multiple devices which incorporate internally ports A, B, and C. The computer has been designed to support up to 256 of these ports so we have numbered them using the hexadecimal numbering system. The possible port numbers can then range from 00 to FF. It will be found that a common characteristic of most single board computers is the use of assigning the port addresses of E8, E9, and EA to the on-board 8255 parallel peripheral interface. Therefore, the

first channel of port A would be defined as having an address of E80; the second channel of port B would be E91, and so forth.

III. SELECTING THE COMPUTER BOARDS

To this point we have delayed the selection of the boards which will be required to provide the computerized control system. The Intel OEM Microcomputer Systems Configuration Guide has been designed to simplify the task of selecting the required system. Our first task is to enter all known information describing our desired system into the project configuration worksheets. These worksheets can then be used to actually select a board configuration which meets our particular requirements. The effort required to accomplish the entry of data is reduced to a minimum through the use of predefined digital and analog configuration worksheets. Our requirement of having a total of 24 parallel data lines, consisting of a mix of high and low level interfaces, can be met by the 24-bit AC/DC combination. Our assignments of requirements for the terminator panels can be made and is shown in Figure 23. It can clearly be seen from the worksheets, that our required interface with the computer digital data will consist of one 24-bit wide connector (had we not used port C assignments, the use of 16-bit wide connectors would have sufficed). This means that our selected single board computer or I/O expansion board must provide at least one edge connector having 24 I/O bits on it.

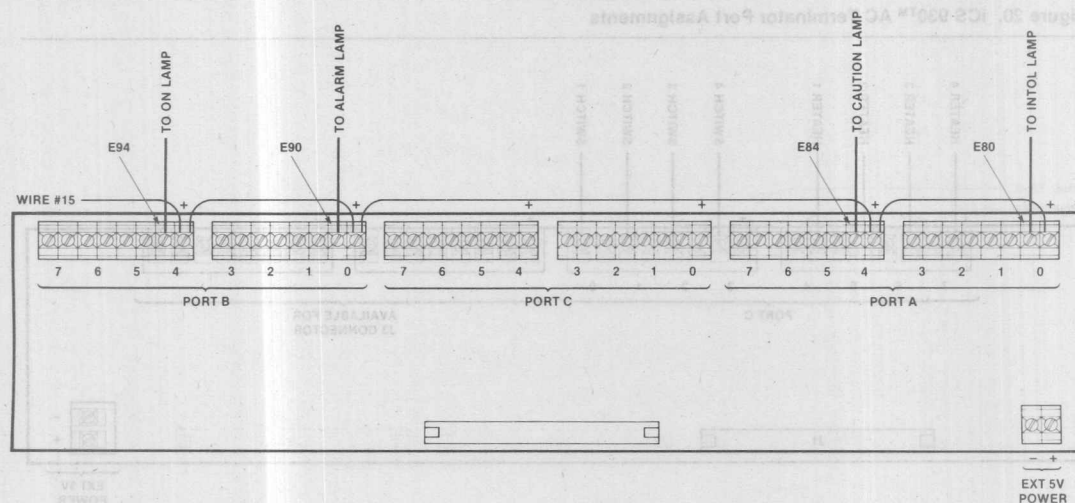


Figure 22. Digital Panel Application Configuration

DIGITAL CONFIGURATION WORKSHEET

PROJECT

This worksheet will provide the required digital interface configuration data which is required to complete the Project Configuration Worksheet.

Enter Number of Channels

Enter # of Discrete AC Outputs (115-230 VAC) 4 (A)
 Enter # of Discrete AC Inputs (115-230 VAC) 4 (B)
 Enter # of Discrete DC Outputs (Current > 300 MA) 0 (C)
 Enter # of Discrete DC Outputs (Current < 300 MA) 16 (D)
 Enter # of Discrete DC Inputs 0 (E)

Compute the Number of iCS 920™ and iCS 930™ Termination Panels

First compute the number of Parallel I/O ports (8-bits each port) required on your iSBC™ board. Round all computations up to the nearest whole integer unless instructed otherwise!

Compute # of iCS 930 Interface Output Ports ((A + C)/8) 1 (F)
 Compute # of iCS 930 Interface Input Ports (B/8) 1 (G)
 Compute # of iCS 930 Termination Panels ((F + G)/2) 1 (H)
 Compute # of iCS 920 Interface Output Ports (D/8) 2 (J)
 Compute # of iCS 920 Interface Input Ports (E/8) 0 (K)
 Compute # of iCS 920 Termination Panels ((J + K)/3) 1 (L)

Optimization of Digital I/O Port Usage for Minimum I/O Configuration

Compute # of iCS 930 Output "Overflow Channels" (DO NOT ROUND OFF)
 (A + C)/8 QUOTIENT 0 (M)
 (Overflow Channels) REMAINDER 4 (N)
 Compute # of iCS 930 Input Overflow Channels (DO NOT ROUND OFF)
 (B/8) QUOTIENT 0 (P)
 REMAINDER 4 (R)
 Compute # of iCS 920 Output Overflow Channels (DO NOT ROUND OFF)
 (D/8) QUOTIENT 2 (S)
 REMAINDER 0 (T)
 Compute # of iCS 920 Input Overflow Channels (DO NOT ROUND OFF)
 (E/8) QUOTIENT 0 (V)
 REMAINDER 0 (W)
 Compute 8-Bit Input Ports Required (P + V) 0 (X)
 Compute 8-Bit Output Ports Required (M + S) 2 (Y)
 Compute 4-Bit Output Ports Required ((N + T)/4) (ROUND UP) 1 (Z)
 Compute 4-Bit Input Ports Required ((R + W)/4) (ROUND UP) 1 (AA)
 Compute 8-Bit Port C Requirements ((Z + AA)/2) (ROUND UP) 1 (BB)
 Total I/O Parallel Ports Required (X + Y + BB) 3 (CC)
 Total # of 24 Channel Parallel I/O iSBC Board Edge Connectors
 (CC/3) (ROUND UP TO INTEGER) 1 (DD)

Compute Power Requirements for the Termination Boards (DO NOT ROUND OFF)

Compute +5V for iCS 920 Board Outputs (.061 * D) 976 (EE)
 Compute +5V for iCS 920 Board Inputs (.023 * E) 0 (FF)
 Compute +5V for iCS 930 Board Outputs (.020 * (A + C)) 080 (GG)
 Compute +5V for iCS 930 Board Inputs (.012 * B) 048 (HH)
 Compute iCS 920 Power Requirements (EE + FF) 976 (JJ)
 Compute iCS 930 Power Requirements (GG + HH) 123 (KK)

Enter the appropriate data into the Project Configuration Worksheet as shown below:

PROJECT CONFIGURATION WORKSHEET

SEE INSTRUCTION SHEET

EQUIPMENT PARAMETERS:

EQUIPMENT	PRODUCT	MEMORY		SERIAL I/O		PARALLEL INPUT/OUTPUT				ANALOG INPUT/OUTPUT				POWER REQUIREMENTS				COST						
		RAM (BYTES)	EPROM (BYTES)	Serial Ports	TTY	RS232C	IN	OUT	IO	24	SI	LOW	IO	VO	IN	OUT	-5V	-12V	5V	-12V	LOW	5V	5V	5V
(L) ea	iCS-920						(E)	(D)										(JJ)						
(H) ea	iCS-930						(B)	(A+C)	(DD)									(KK)						
TOTAL																								
TOTAL																								

Figure 23. Digital Configuration Worksheet

[illegible]

A similar technique is used to configure the analog signals using the standard analog configuration worksheet as shown in Figure 25. It can be seen that our application will require a single cable connection to a differential input edge connector of an analog input board. The power requirements can be calculated from the current requirements to drive the thermistors and the sensing resistors. The data is entered into the appropriate columns of the configuration tables and then transferred to the project configuration worksheet.

PROJECT OVER CONTROLLER

Enter Number of Channels

Compute the Number of ISBC™ Board Edge Connectors

Compute the Number of ICS-910™ Termination Panels

Place the appropriate data into the Project Configuration Worksheet as shown below:

PROJECT CONFIGURATION WORKSHEET

[illegible]

The only remaining physical element of our control system which we have not defined is the CRT terminal which will be used for setpoint entry and modification. Communications with a terminal requires that we provide a serial RS232C port in our control system. This port requirement is entered

[illegible]

We must now choose the Intel iSBC boards which will provide a solution to our system requirements. This is done by referencing the summary of key iSBC configuration parameters to find boards which provide the necessary characteristics. Our first task is to choose a single board computer which meets as many of our needs as is practical, while providing performance characteristics adequate to our needs.

Our first requirement for having support for a single RS232C serial communications channel can be seen to be met by a variety of possible boards. Among the possible boards meeting this requirement are:

iSBC 86/12™ iSBC 80/10A™
iSBC 80/20™ iSBC 80/20-4™
iSBC 80/30™

10-36

[illegible]

Since our application is not likely to require extensive math handling capabilities or high speed capabilities, we probably do not need the power found in the iSBC 86/12; so we will remove this product from consideration.

iSBC 80/20 board advantages — Slightly lower cost, greater number of I/O lines available.

iSBC 80/30 board advantages — Faster processor, dual ported memory, able to utilize UPI modules.

A good design practice is to provide an extra margin of available memory in the hardware design. Our anticipated RAM memory will use about 2K bytes. The computer will provide us with 4K bytes so we have a considerable margin. This is not true when we look at the amount of EPROM available on the board. Our 8K requirement is identical to

The computer selection and the memory expansion board data can now be entered onto the configuration worksheet as shown in Figure 28. If needed, the addition of the memory expansion board will allow our EPROM requirements to grow up to 16K bytes.

[illegible]

The only requirement which we have not met is to assign a board to handle the analog input needs of our temperature sensing circuit. The analog voltage can be calculated and will be found to lie in the neighborhood of 4.6 volts at room temperature. This value will increase toward 5 volts as the temperature of the oven increases. Since we have no requirement for any analog output capabilities, we will choose the Intel® iSBC 711™ Analog Input Board to sense the voltage level. This board can be configured to handle a 5-volt full scale input and will provide a resolution of 12 bits. (If an oven requiring a wide range of temperatures and greater resolution were required, we would have to reconfigure our temperature sensor to provide a wider voltage spread over operating temperatures. For purposes of simplicity and clarity we will assume that our temperature resolution is adequate.)

The Industrial Chassis

Before the boards can be operated together to form a control system, a means of allowing communica-

SEE INSTRUCTION SHEET

PROJECT CONFIGURATION WORKSHEET

EQUIPMENT PARAMETERS:

YOUR REQUIREMENTS	PRODUCT	MEMORY		SERIAL I/O		PARALLEL INPUT/OUTPUT		ANALOG INPUT/OUTPUT		POWER REQUIREMENTS				COST					
		RAM (Kb)	EPROM (Kb)	TTY	RS232C	IN	OUT	IO	24	DI	DO	LOW	HIGH	5V	12V	LINE	10V	0V	0V
	1ea. iCS-910									4	0	0	0	0	1				
	1ea. iCS-920					0	16							.976					
	1ea. iCS-930					4	4	1						.12%					
	CRT					1													
	TOTAL		2K	5K		1	4	20	1	4			1	1.104					
INTEL SOLUTION	SLOT #1 iSBC 80/30	16K	5K			1	4	20	1	1				3.500	.220	.0025	.050		
	SLOT #2 iSBC 416		16K											.750					
	SLOT #3 iSBC 711								8			1	1.700						
	SLOT #4																		
	SLOT #5																		
	SLOT #6																		
	SLOT #7																		
	SLOT #8																		
	SLOT #9																		
	SLOT #10																		
	SLOT #11																		
	SLOT #12																		
	ROM (Kb)													.090					
	I/O DRIVERS													.420					
	TERMINATION																		
	USER SUPP. HARDWARE																		
	SUBTOTAL													3.666	.220	.0025	.050		

SYSTEM INTEGRATION	SLOTS AVAILABLE	POWER AVAILABLE	-5V	12V
SYSTEM				
CHASSIS: iCS 80-35	4	14.0	2.0	900
CARD CAGE				
POWER SUPPLY				
TOTAL SYSTEM COST				

NOTE:
 DI: Differential Input
 DO: Digital Output Input
 IO: Current Output
 VO: Voltage Output

Figure 29.

tion between the boards and of distributing power among the boards must be found. This requirement is met by specifying a chassis into which the boards will be mounted. The Intel® iCS 80™ Industrial Chassis provides an environment for operating the boards which is specifically designed to operate in an industrial area.

The chassis has been designed to facilitate mounting into either a standard 19-inch RETMA cabinet or it may be rear-panel mounted into an enclosure such as may be found in applications requiring the use of a NEMA electrical enclosure. The card chassis has been mounted in such a manner as to hold the single board computers and expansion modules vertically, facilitating maximum cooling of the boards. Fans are provided to aid the normal convection cooling process. Card racks may be installed into the iCS 80 chassis to expand the card support capability to a maximum of 12 card slots in groups of four. Either an iSBC 635 or 640 power supply can be mounted into the industrial chassis to provide power up to 4 or 12 boards capability, respectively.

Our application design requires the installation of a three board solution, so we will choose the iCS 80 chassis with one iSBC 635™ power supply. We will choose to mount our control system in a standard NEMA 12 enclosure to protect the unit from the industrial environment. We should refer to the *iCS 80 Industrial System Site Planning and Installation Guide* (manual order number 9800798) for complete details for selecting appropriate enclosures and installation instructions.

The +5 volt power needed to support the various termination panels and to supply a reference voltage for the thermistors is available from a barrier strip located on the lower front of the iCS 80 chassis (Figure 30). Our wiring can be routed to this barrier strip for those circuits requiring either 5-volt DC or the system logic common. A fuse holder is provided and a fuse should be installed for system protection. We will install a 2-amp fuse into the holder (our maximum power requirement for external circuitry should be 1.22 amps according to Figure 26).

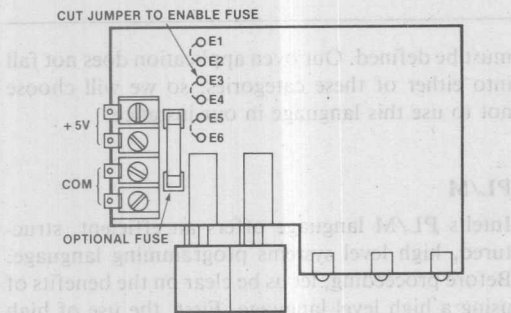


Figure 30. Industrial Chassis DC Power Strip

The remaining terms required in our ladder diagram (Appendix B) consist of a high voltage neutral and a source of switched high voltage power for the heater lamps. Both of these terms are available from the iCS 80 industrial chassis. It is desirable to utilize the same switched power for both the computer system and our external signals, so that we can provide protection to operators when one portion of the system is shut down. A common source will insure that all portions of the system are inactivated if repair is being done. The iCS 80 chassis incorporates a heavy duty industrial key-lock switch for its power switching. The outputs of this switch are available to the user at a terminal barrier strip located on a fold-out panel on the rear of the chassis assembly (refer to Figure 31). We can see that our neutral wire should be connected to terminal 5 (filtered AC low) and the wire for the AC high, wire #10 on the ladder diagram, should be connected to terminal 9. This will provide us with a switched, fused, and filtered power source for our external wiring.

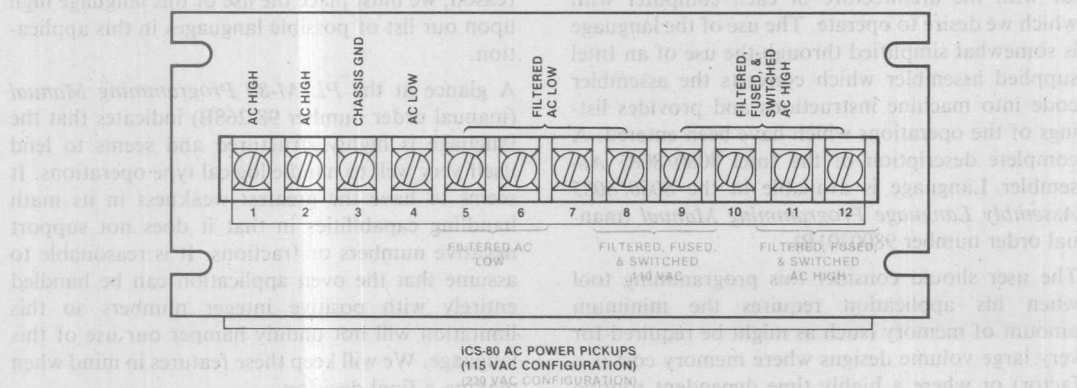


Figure 31. Industrial Chassis AC Power Strip

As we will be installing the chassis into a NEMA enclosure, we will not want to use a standard power cord since this would involve the additional expense of installing a duplex outlet in the cabinet. The power wiring can be installed directly onto the power barrier strip by placing the AC hot wire on barrier number 1, the neutral wire onto barrier number 4, and the ground onto barrier number 3.

The hardware implementation of the system can now be considered to be complete. Before the system can function as a control for the oven temperatures, we must define the relationships between the various pieces of the oven system and we must also define the operator interface with the CRT terminal. Thus, we begin the software phase of our design.

IV. DETERMINATION OF SOFTWARE APPROACH

The task of providing the relationships between the various system components falls into the category of writing the software. Before we actually begin to develop this software, we will define certain guidelines which can be used to organize and simplify the task.

Let us consider the general environment under which our programs will operate. We find that we have essentially two choices in this area. First, we can consider the entire process as a sequential set of predefined operations in which we must perform each operation before moving to the next until finally we complete the sequence and begin again. (This is analogous to using a single stepper switch to design our control system.) Since each oven is independent of the others, we can not afford to use

this approach since we could get tied up waiting for something to happen in a particular oven and would have to ignore the other ovens. The designer familiar with relay design will probably be thinking, at this point, that we should use a separate sequential operation for each oven or device to be controlled. Indeed, this is exactly what we can do with our software by using what is known as a real-time executive. This tool will allocate the computer's resources in such a manner as to provide us with the capability of having independent software programs or tasks operating at what appears to be the same time. We will make our first assumption that our software will be written using such a tool and we will specify that we will operate under Intel's RMX/80 Real-Time Multi-Tasking Executive. We will discuss more detail of this software tool as we develop our programs.

Next, we must consider the language which we will use to actually define our required operation. We have many alternatives from which to choose. Let us look at several of the alternatives in some detail.

Assembler

Assembler language is probably the most basic tool with which we can program a computer. It is considered to be the most efficient user of program memory and processor time. These features are made possible because each assembler instruction line is converted directly into a corresponding machine instruction. From a programming standpoint, assembler language is the most difficult to use since any task must be defined by subdividing that task into a multitude of smaller operations compatible with the available instructions of the computer. To use this language, we must be familiar with the architecture of each computer with which we desire to operate. The use of the language is somewhat simplified through the use of an Intel supplied assembler which converts the assembler code into machine instructions and provides listings of the operations which have been entered. A complete description of the Intel 8080/8085 Assembler Language is available in the *8080/8085 Assembly Language Programming Manual* (manual order number 9800301B).

The user should consider this programming tool when his application requires the minimum amount of memory (such as might be required for very large volume designs where memory cost is a factor) or where a highly time dependent routine

must be defined. Our oven application does not fall into either of these categories, so we will choose not to use this language in our instance.

PL/M

Intel's PL/M language offers an efficient, structured, high level systems programming language. Before proceeding, let us be clear on the benefits of using a high level language. First, the use of high level languages results in reduced development time and cost. High level languages provide the ability to program in a natural algorithmic language. In addition, they eliminate the need to manage register usage or to allocate memory. Second, high level languages provide improved product reliability because programs tend to be written in structured formats and result in a minimum of extraneous branches which might cause testing problems. Finally, their use produces programs which are better documented and are easier to maintain.

On the other hand, high level languages do not optimize the code segments as well as can be done by an experienced assembly language programmer. As a result, most compilers (routines which convert the high level languages into machine executable code) use more program storage than those written by the assembly language programmer. Different languages and compilers require different amounts of memory for the same task.

PL/M-80 is probably one of the most efficient high level languages for use on microcomputers. It has been determined that PL/M-80 users can expect to use between 1.1 to slightly more than 2 times as much program memory as would be used for the same task written in assembly language. For this reason, we must place the use of this language high upon our list of possible languages in this application.

A glance at the *PL/M-80 Programming Manual* (manual order number 98-268B) indicates that the language is highly structured and seems to lend itself very well to handle logical type operations. It seems to have the greatest weakness in its math handling capabilities in that it does not support negative numbers or fractions. It is reasonable to assume that the oven application can be handled entirely with positive integer numbers so this limitation will not unduly hamper our use of this language. We will keep these features in mind when making a final decision.

FORTRAN

Intel's FORTRAN-80 provides the full subset of ANSI FORTRAN 77. In many cases FORTRAN-80 has features that exceed the specifications for both the subset and the full versions of FORTRAN 77. Most of the power of this language lies in its ability to easily handle complex mathematical expressions. Obviously, it does not have any limitations regarding fractions or sign of the numbers involved. It should be used when the application requires the use of mathematical computations. The power of the language, however, means that the use of the language will take a heavy toll of memory allocation. A complete description of the FORTRAN version supported by Intel and its use on the iSBC computers can be found in the *FORTRAN-80 Programming Manual* (order number 9800481A) and in the *ISIS-II FORTRAN-80 Compiler Operator's Manual* (order number 9800480).

It is unlikely that the magnitude of mathematical routines required to control the temperature of our ovens will be complex enough to justify the use of FORTRAN. Keep in mind that, if such a situation were encountered, it is feasible to use a combination of programming languages to create our final module.

BASIC

Certainly the most well known high level programming language today is BASIC. It offers a quick way of applying the computational capabilities of the computer to a wide range of applications. The Intel RMX/80 BASIC-80 is an interpreter designed to operate with Intel's single board computers and contains extended disk handling capabilities. As an interpreter, it differs from other high level languages in that it results in a relatively slower operating solution to an application. It is also not possible to use BASIC to generate multiple independent tasks which can compete for computer resources. For these reasons, we cannot consider the use of BASIC for a solution to our application.

Final Selection of Language

From the above discussion, it seems clear that our choice for the application being demonstrated is to use PL/M-80 as our programming language.

With this in mind, we can begin the task of actually generating the code which will complete our application and provide an operating control system.

V. DEFINING SOFTWARE TASKS

The software implementation can begin as soon as we have broken our control functions into independent "tasks". We can then handle each task separately as though it were the only thing which had to be done by the control system. In the event that we find that one of our tasks must communicate with or be interlocked with another, we will handle this need through the use of "exchanges". The "exchange" can be thought of as a mailbox into which messages are deposited and picked up by the various tasks. These messages convey the necessary information between the otherwise independent programs. When all tasks have been coded, we will combine them using the facilities of RMX/80.

Our oven application can be broken down into three functional areas or tasks. These are:

1. The Control Task which will be used to actually sense the oven temperature and to provide the required responses to the heaters and the indicator lamps.
2. The CRT Update Task will be used to provide a "snapshot" of the system operations to a person viewing the CRT terminal.
3. The Parameter Update Task will be used to examine and update the oven setpoints and tolerances.

The choice of these three tasks has been essentially arbitrary in nature. Certainly, other choices and groupings of functions could easily have been made. We will use these choices for our example and will proceed with our development accordingly.

We have two other supporting tasks which must be included in our system. Fortunately, these tasks are predefined and fully supported within RMX/80's libraries; thus we need not write these functions. The two supporting tasks are:

4. A Terminal Handler Task to support the actual interface to the CRT terminal. It provides echo of input characters and signals when data is ready to be read. It will output messages to the terminal and signal when all characters requested have been sent.
5. An Analog I/O Driver Task to request and handle the handshaking which is required to communicate with the analog input board. It will signal us when data has been input and is available for use by our user written tasks.

We can proceed with the implementation of each of our three tasks which we have defined. The first step with each will be to develop a flowchart which shows the required operations to implement that task. This flowchart will show any intertask communications or exchanges that may be required with other tasks. The flowchart can then be coded using the facilities provided by our programming language.

Oven Control Task

The sequence of operations required to perform the control task can be defined using the flowchart shown in Figure 32. Let us examine the required steps in more detail.

An arbitrary decision has been made to only sample and control the ovens once each second. This will allow some time for the system to respond once a heater output has been set. The first step in our control task is to wait for one second to elapse.

Our next subtask should be to read the status of the various oven control switches on the operator's control panel. This item could wait until a later time, but there is no harm in handling it at this time.

Next, we see a block indicating the input of data regarding the current oven temperatures. This oven temperature data will certainly be used by the task handling the snapshot display on the CRT so we must give some consideration to the validity of the data. While we are in the process of getting the data and converting it to engineering units (next step), there will be periods during which the stored temperature data does not reflect the actual oven temperature. An example might be when we are actually moving the 16 bits of the temperature since we can only move data 8 bits at a time. During this period, we would not want another task to use the data and since each task is going to operate independent of others, we must provide some type of lockout of the data while we are operating on the temperatures (an alternative would be to have each task get its own temperature from the A/D converter and convert it to engineering units, but this would seem to waste memory and computer time). We can provide this lockout by creating an exchange to communicate with other tasks. If we make a message available in this exchange when the data is valid and cause no messages to be available when the data is nonvalid, we can effectively lock out tasks from using the data when it is in the process of being updated. This is done by requiring

those tasks to test for the presence of a message at the exchange before they get the temperature data. If no message is present, they must wait until one is placed into the exchange before proceeding. Just before we update the temperatures we will fetch the message from the exchange, leaving it empty while we work on the data. later we will again restore the message when the update is complete.

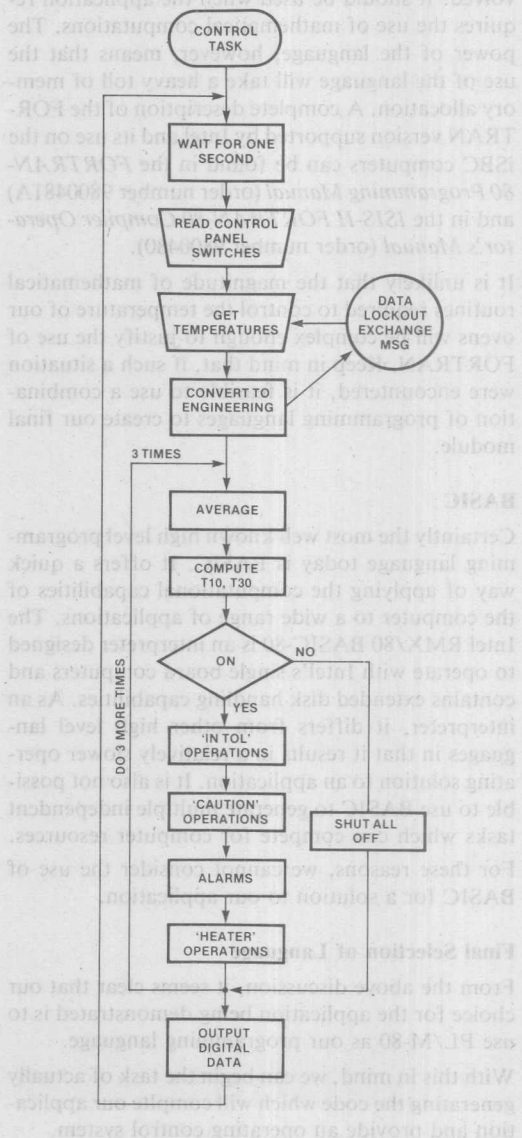


Figure 32. Control Task Flowchart

The number obtained from the analog converter provides us with a value which is proportional to the temperature of the oven. Our next step is to convert this number into engineering units. Unfortunately, the voltage and temperature are not related in a linear fashion since the thermistor is a nonlinear device. We will have to develop a technique to obtain a corrected value. For the purposes of this application note and in an attempt to keep the application as simple as possible, we have chosen to utilize a single table look-up to perform this conversion. Alternatives might have been to utilize FORTRAN routines to mathematically perform the conversion or to have separate tables for each oven. Once the conversion has been made, we must return a message to the data lockout exchange to allow other tasks access to the data.

Because we must deal with four ovens, the operations related to each individual oven must be performed four times, once for each. This is easily handled as we will see, since PL/M is a block structured language. Our flowchart need only remind us that the operations need be done four times.

The next step has been defined as performing some digital filtering of the temperature by averaging the current temperature with the temperature of one second ago. This filtered value will be used to perform subsequent computations and to make future decisions.

We have defined earlier in our definition of the control algorithm that we would use a derivative control. We have chosen to project the temperature ahead for a period of 10 and 30 seconds. We must calculate the rate of change and the temperatures in 10 and 30 seconds so that this data will be available when needed.

Now that the calculations have been made to determine numeric values required for the decision making process, we must begin the process of determining the status of each indicator and oven heater. A test will be made of the oven run switch and if it is found to be turned off, we will turn off all indicators and the oven heater associated with that oven. If the switch is found to be turned on, we will set the status of the "in tolerance", "caution", and "alarm" indicators according to our oven control algorithm. The oven heater will be turned on or off according to the projected temperature in 30 seconds.

Rather than output the individual oven indicator and heater data four times (once for each oven), we

will perform the computations associated with making the decision four times (this saves code since we can use the same program steps with only pointers being exchanged). At the end of this time, a single operation will output the data to all ovens and indicators at the same time. Outputting to a computer port will actually cause the device to turn on or off according to whether the output bit is a one or zero.

We will then return to the beginning of our task to wait until another second elapses before we again perform the indicated functions.

Control Task Source Coding — The coding of our tasks is a straightforward procedure once we have prepared a flowchart. Since we are using PL/M-80 and RMX/80, the coding sequence for a task will be as follows:

1. Define any variables or structures which will be used in the module. This involves providing information defining variables as being either an 8 or 16-bit variable and declaring if that variable is to be a part of the task being coded or is to be found in some other task. If any arrays or structures are to be used, they must also be defined. Finally, if any program locations are to be used, they must be declared.
2. The task must be initialized. That is to say that any assumptions which will be made as to initial data values in subsequent instructions must be initially forced to this initial value.
3. The actual task must be coded to match the operations called out in the flowchart.

We will look at some examples of this coding process using the control task flowchart. The complete listing of this module and all modules actually used to provide the oven control system can be found in Appendix C.

At first glance, it would seem that the listing is extremely complex, but as we will see it is made up of straightforward pieces. The listing is made up of three parts as we have mentioned above when defining the steps required to generate a program. The first part (line numbers 1 through 50) is used to define parameters, variables, and external elements. The general types of elements making up this portion fall into typical categories. The first general category consists of DECLARE statements. Examples of typical lines will help explain their meanings (when actually developing the program, this first section was created piecemeal by

making an entry when it was found that a need for that term existed as the execution code in sections two and three were written).

Examples of the "declare" statement are shown below. For example, on line 11 we find:

```
11 1 Declare (n,k) byte;
```

This means that the variables "n" and "k" are being defined as terms which represent numbers or data which is one byte or 8 bits wide. The "11" is the program line number, and the "1" indicates that we are in the first level of nesting.

We can also see the use of the "literal" expressions such as used in line 4. The expression:

```
4 1 DECLARE FALSE LITERALLY '00H';
```

means that we are creating a new instruction called "false" and that its meaning is to be interpreted by the compiler as being equivalent to the value of zero.

Rather than dwell on the declaration, let us move on to the coding process which was used to generate the actual program. Keep in mind that the use of PL/M-80 requires that all terms used be declared in the program module. Refer to the *PL/M-80 Programming Manual* (order number 9800268B) for a full description of the PL/M language.

Program Initialization — The initialization portion of the program can be found on lines 51 through 59 of the control task program listing. This section is used to initialize data and to provide known entry conditions before we enter the repetitive program loop. This code is only executed when the system is reset or when the power is turned on. The control task requires two types of initializations; one to initialize the computer's output port and the other to set up the A/D converter. The requirements for each can be found in the RMX/80 User's Guide and the *iSBC 80/30 Single Board Computer Hardware Reference Manual* (order number 9800611A). Actual instruction examples are given in these manuals for the initialization operations.

Program Body — The program which actually provides the control operations can be found on lines 60 through 126 of the program listing for the control task. It has been divided into sections which correspond directly to the flowchart that was prepared earlier. Most instructions in PL/M-80 language follow closely the English structure which describes what is being done. The exceptions generally follow definite predefined formats. The for-

mat such as used on line 61 to wait for one second to elapse is an example of one such exception. Any time we desire to wait for a definite time period, we use an instruction of the form:

```
MSG$PTR=RQWAIT (.DUMMY$EXCH, TIME DELAY);
```

Whatever time delay we wish to use is expressed in increments of 50 msec time periods. Our example requires a time delay of one second so we will use the delay notation of $1.0/0.050 = 20$ time units (this command is actually calling upon the RMX/80 executive to handle the delay).

The oven enable switch data has been defined by us to be routed by the hardware to the computer port "EA" which converts to a decimal number, 234. If we define an internal memory location for this data and call it BLOCK0, then we can get the oven switch data by using an input statement. Since the data sense is inverted through the hardware, we can provide meaningful internal data if the signal is re-inverted as it is loaded into memory. The instruction on line 62 of the control task listing performs this task.

We are now ready to get the analog data from the A/D converter. Our flowchart shows that we must lock out the other tasks from access to the temperature data during this time period, so we must first remove the enable message from the exchange in which it is stored. Messages are removed from an exchange by using an instruction of the form:

```
STORAGE=RQWAIT (EXCHANGE NAME,0)
```

Line 63 of the program listing means that we will get a message from our storage exchange which is called "Temp\$lockout\$exch" and store it in a memory storage area called "Lockout". Now, no other task can get a message from this exchange since it is empty, so it is permissible to operate on the temperature data. (Note how similar this command is to the one used to wait for a delay. Indeed, this is the same request for RMX/80, but it requests a time delay of zero.)

During the initialization, we built a message defining the characteristics of the analog signals and of the analog conversion board which we are using. Remember that we have indicated that the task of getting this data from the board is provided to us by one of RMX/80's predefined drivers. All that is necessary at this time is to inform that driver of our desire to get data, then wait until it has done its job and the data is available for us. The actual communication between our applications task and the analog driver is done using the idea of an exchange similar to that we have used to lockout the data.

A flowchart can now be prepared showing the steps required to implement the CRT update task. This flowchart is shown in Figure 34. The coding of the program to support this task can be found in Appendix C. The development is identical with that which we described in the sections regarding the control task. Again, the software is divided into three parts, the declaration statements from lines 1 to 81, the initialization on lines 82 to 87, and the actual task code on lines 88 to 207.

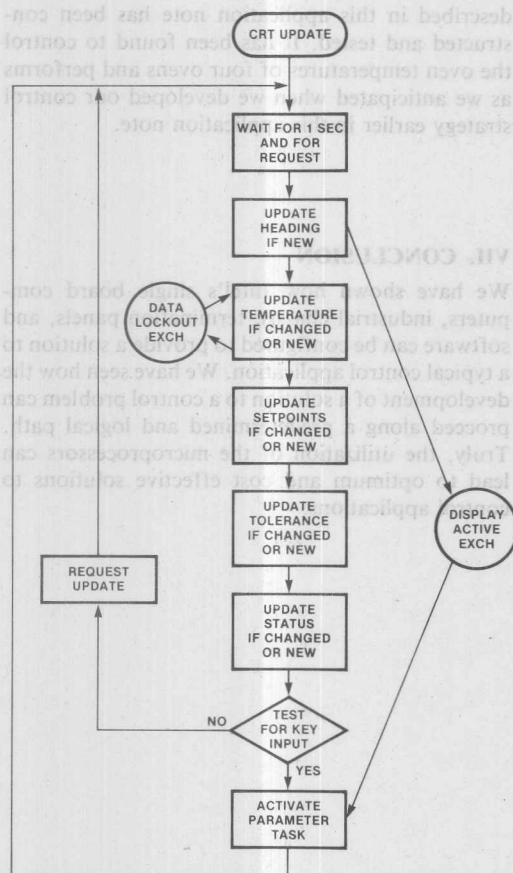


Figure 34. CRT Status Flowchart

A technique to exit from the CRT update mode and to get into a mode which will allow modification of the parameters has been introduced into the program and the display format. This is in the form of a message on the bottom of the screen requesting the entry of an escape character to adjust setpoints. The software has been written in such a

manner as to test for a character input from the keyboard and if one is found corresponding to that character, the update task will allow the parameter update task to take control of the terminal (lines 190 to 204 of the listing).

Parameter Update Task

The parameter update task is used to actually allow the modification of the setpoints and the tolerances associated with each oven. A second use of the task is to provide a tool for establishing the zero offset associated with each analog channel so that an offset into the temperature linearization table can be computed by the control task.

Figure 35 shows the flowchart which describes the steps required to perform these operations. When the task has been completed, we will return to the CRT update task.

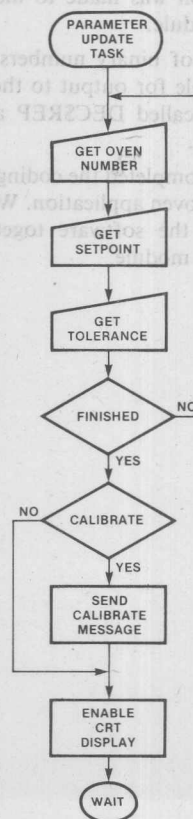


Figure 35. Parameter Update Flowchart

The program code for this task can be found in Appendix C and again follows the formats which we have discussed earlier. No attempt will be made in this document to provide a narrative of the listing since it follows the flowchart in development.

Support Programs

Three subprograms (procedures) have been written which provide functions which are common to the three tasks. This has been done to minimize repeating code segments thus saving as much memory as possible. These three subprograms support:

1. Conversion of a decimal string from the terminal into a binary number. This program is called ASC\$2\$BINARY and can be found in Appendix C.
2. Storage for common variables used by more than one task. These variables could easily have been included in other tasks but a purely arbitrary decision was made to include them in a separate module.
3. Conversion of binary numbers into a decimal string suitable for output to the terminal. This program is called DEC\$REP and is found in Appendix C.

We now have completed the coding of the software to support our oven application. We must finish by combining all the software together to form a single loadable module.



Figure 35. Parameter Update Flowchart

VI. FINAL IMPLEMENTATION

When all code was linked and loaded to form an executable program module, it was found that the system required 9,041 bytes of EPROM and 1,735 bytes of RAM. These values fall within our hardware capabilities and will require that we program and insert nine EPROMs into the EPROM expansion card.

The system can now be tested and installed to control the ovens of our application. The actual system described in this application note has been constructed and tested. It has been found to control the oven temperatures of four ovens and performs as we anticipated when we developed our control strategy earlier in this application note.

VII. CONCLUSION

We have shown how Intel's single board computers, industrial chassis, termination panels, and software can be configured to provide a solution to a typical control application. We have seen how the development of a solution to a control problem can proceed along a predetermined and logical path. Truly, the utilization of the microprocessors can lead to optimum and cost-effective solutions to control applications.

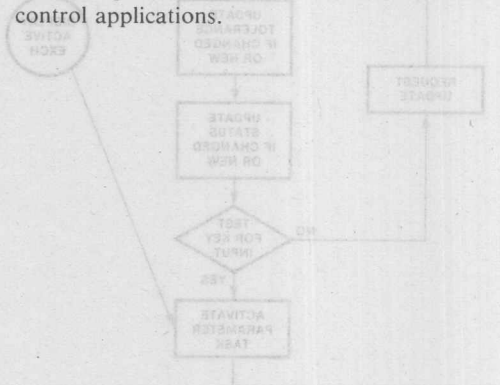


Figure 36. CRT Status Flowchart

A technique to exit from the CRT update mode and to get into a mode which will allow modification of the parameters has been introduced into the program and the display format. This is in the form of a message on the bottom of the screen requesting the entry of an escape character to adjust response. The software has been written in such a

We will send a message to the analog driver telling it what we want it to do, then we will wait until it sends a message back to one of our exchanges telling us that it is done. The format for sending a message to an exchange always follows the form:

CALL RQSEND (EXCHANGE NAME, MESSAGE NAME);

Line 64 of the listing shows that we have requested the input of the analog data since we have sent our message, Convert, to the analog driver's exchange which is called RQAIEX. We will wait until the operation is complete by using the line of code shown on the listing line 65. This is the same operation type that we used to get our message back providing a lockout earlier. The program will wait until a message is available before continuing.

The data must now be converted into engineering units. We earlier indicated that we would use a table lookup to perform the linearization, so we have included this table as a part of our program at line 50. The offset into the table corresponding to our temperature must be determined so that the correct value can be stored. Because we have four ovens, we will perform the operation four times with the data each time corresponding to the appropriate oven. These operations can be followed on lines 77 through 81 of the listing.

Lines 67 through 76 are used to establish an offset to be applied to the analog temperature data when the system is running. This program is only designed to be used during the start-up operations and is activated when a message containing a calibration request and current temperature is sent to its exchange.

The temperature lockout must be removed to enable other tasks to use this data. This is done on line 82 by sending the message back to the exchange used for intertask lockout communications.

The remainder of the program follows the flowchart and the operations can be followed using a flowchart and the listing. Each element of the flowchart corresponds to a block of code on the listing.

CRT Update Task Development

Earlier, we stated that the CRT update task would be used to allow the operator to view a "snapshot" of the four ovens. Let us turn our attention to developing the software which is required to accomplish this. We can begin by defining the elements which we feel should be displayed, then defining the format to actually be used with the CRT terminal.

Obviously, we need to provide the current temperature of each oven on our display screen. If we display the actual temperature, it seems reasonable to assume that we should also show the setpoint so that a determination can be made as to how well the system is performing. The control algorithm has been defined to use an allowable range to determine system outputs, so it would seem wise to also show this parameter. Finally, we should inform the viewer of the status of the oven so that he will realize that the reason an oven temperature is low is because the oven is off rather than an oven malfunction. Other items could be added if desired by the system designer, depending upon the total system requirements or the characteristics of the users.

We can now prepare a drawing of the CRT display to generate a layout of our desired characters and to generate an aesthetic display for viewing during operation. This drawing can be found in Figure 33.

Several techniques are available to output the required displays to the terminal. A decision must be made as to the frequency of screen updates; will we constantly refresh the data or do it only at certain intervals of time? If the terminal has the ability to disable the cursor, it makes sense to update data continuously. If the cursor cannot be disabled, its movement tends to be distracting, so the updates should be kept to a minimum. The terminal used for the application note did not have a disable feature, so we will make the decision to only update the screen once each second.

OVEN STATUS DISPLAY					
	OVEN-1	OVEN-2	OVEN-3	OVEN-4	
TEMPERATURE	XXX.X	XXX.X	XXX.X	XXX.X	DEGREES
SETPPOINT	XXX.X	XXX.X	XXX.X	XXX.X	DEGREES
TOLERANCE	XXX.X	XXX.X	XXX.X	XXX.X	DEGREES
STATUS	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	

TYPE ESCAPE TO ADJUST SETPOINTS.

Figure 33. CRT Status Display Layout

The decision to delay updates leads us to make another decision regarding the screen updates. If we only update a line which has data which has changed since the last update, the cursor movements will be kept at a minimum since it is unlikely that all parameters will ever change each second.

Obviously, we need to provide the current temperature of each oven on our display screen. If we display the actual temperature, it seems reasonable to assume that we should also show the setpoint so that a determination can be made as to how well the system is performing. The control algorithm has been defined to use an allowable range to determine system output, so it would seem wise to also show this parameter. Finally, we should inform the viewer of the sign of the error so that he will realize that the reason an oven temperature is low is because the oven is off rather than an over-maintenance function. Other items could be added by the system designer, depending upon the total system requirements or the characteristics of the users.

We can now prepare a drawing of the CRT display to generate a layout of our desired characters and to generate an aesthetic display for viewing during operation. This drawing can be found in Figure 33.

Several techniques are available to output the required displays to the terminal. A decision must be made as to the frequency of screen updates; will we constantly refresh the data or do it only at certain intervals of time? If the terminal has the ability to disable the cursor, it makes sense to update data continuously. If the cursor cannot be disabled, the updates should be kept to a minimum. The terminal used for the application note did not have a disable feature, so we will make the decision to only update the screen once each second.

TEMPERATURE	PRESSURE	STATUS	...
XXXX	XXXX	XXXX	XXXX
XXXX	XXXX	XXXX	XXXX
XXXX	XXXX	XXXX	XXXX
XXXX	XXXX	XXXX	XXXX

Figure 33. CRT Status Display Layout

The decision to delay updates tends to make another decision regarding the screen updates. If we only update a line which has data which has changed since the last update, the cursor movement will be kept at a minimum since it is unlikely that all parameters will ever change each second.

We will send a message to the analog driver telling it what we want it to do, then we will wait until it sends a message back to one of our exchanges telling us that it is done. The format for sending a message to an exchange always follows the format: CALL SEND EXCHANGE NAME, MESSAGE NAME. Line 84 of the listing shows that we have requested the input of the analog data since we have sent our message. Convert to the analog driver's exchange which is called RQAIBX. We will wait until the operation is complete by using the line of code shown on the listing line 85. This is the same operation type that we used to get our message back providing a lockout earlier. The program will wait until a message is available before continuing.

The data must now be converted into engineering units. We earlier indicated that we would use a table lookup to perform the translation, so we have included this table as a part of our program in line 86. The offset into the table corresponding to our temperature must be determined so that the correct value can be stored. Because we have four ovens, we will perform the operation four times with the data each time corresponding to the appropriate oven. These operations can be followed on lines 87 through 91 of the listing.

Lines 92 through 96 are used to establish a table to be applied to the analog temperature data which the system is running. This program is only designed to be used during the start-up operations and is activated when a message containing a calibration request and current temperature is sent to its exchange.

The temperature lockout must be removed to enable other tasks to use this data. This is done on line 97 by sending the message back to the exchange used for intertask lockout communication. The remainder of the program follows the flowchart and the operations can be followed using the flowchart and the listing. Each element of the flowchart corresponds to a block of code on the listing.

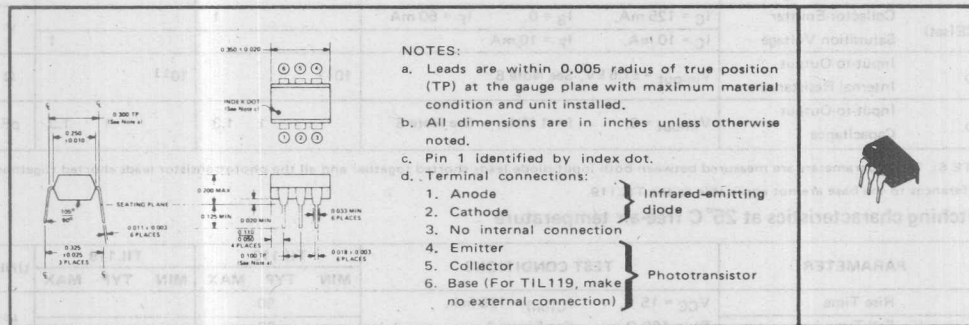
CRT Update Task Development

Earlier, we stated that the CRT update task would be used to allow the operator to view a "snapshot" of the four ovens. Let us turn our attention to developing the software which is required to accomplish this. We can begin by defining the elements which we feel should be displayed, then defining the format to actually be used with the CRT terminal.

- Gallium Arsenide Diode Infrared Source Optically Coupled to a Silicon N-P-N Darlington-Connected Phototransistor
- High Direct-Current Transfer Ratio . . . 300% Minimum at 10 mA
- Base Lead Provided for Conventional Transistor Biasing
- High-Voltage Electrical Isolation . . . 1500-Volt Rating
- Plastic Dual-In-Line Package
- Typical Applications Include Remote Terminal Isolation, SCR and Triac Triggers, Mechanical Relays, and Pulse Transformers

mechanical data

The package consists of a gallium arsenide infrared-emitting diode and an n-p-n silicon darlington-connected phototransistor mounted on a 6-lead frame encapsulated within an electrically nonconductive plastic compound. The case will withstand soldering temperature with no deformation and device performance characteristics remain stable when operated in high humidity conditions. Unit weight is approximately 0.52 grams.



absolute maximum ratings at 25°C free-air temperature (unless otherwise noted)

Input-to-Output Voltage	±1.5 kV
Collector-Base Voltage (TIL113)	30 V
Collector-Emitter Voltage (See Note 1)	30 V
Emitter-Collector Voltage	7 V
Emitter-Base Voltage (TIL113)	7 V
Input-Diode Reverse Voltage	3 V
Input-Diode Continuous Forward Current at (or below) 25°C Free-Air Temperature (See Note 2)	100 mA
Continuous Power Dissipation at (or below) 25°C Free-Air Temperature:	
Infrared-Emitting Diode (See Note 3)	150 mW
Phototransistor (See Note 4)	150 mW
Total (Infrared-Emitting Diode plus Phototransistor, See Note 5)	250 mW
Storage Temperature Range	-55°C to 150°C
Lead Temperature 1/16 Inch from Case for 10 Seconds	260°C

NOTES: 1. This value applies when the base-emitter diode is open-circuited.

2. Derate linearly to 100°C free-air temperature at the rate of 1.33 mW/°C.

3. Derate linearly to 100°C free-air temperature at the rate of 2 mW/°C.

4. Derate linearly to 100°C free-air temperature at the rate of 2 mW/°C.

5. Derate linearly to 100°C free-air temperature at the rate of 3.33 mW/°C.

TEXAS INSTRUMENTS
INCORPORATED

POST OFFICE BOX 5012 • DALLAS, TEXAS 75222

Reprinted with permission from Texas Instruments, March, 1979. All rights reserved.

TYPES TIL113, TIL119

OPTO-COUPLEDERS

electrical characteristics at 25°C free-air temperature

PARAMETER	TEST CONDITIONS†	TIL113			TIL119			UNIT
		MIN	TYP	MAX	MIN	TYP	MAX	
$V_{(BR)CBO}$ Collector-Base Breakdown Voltage	$I_C = 10 \mu A$, $I_E = 0$, $I_F = 0$	30						V
$V_{(BR)CEO}$ Collector-Emitter Breakdown Voltage	$I_C = 1 mA$, $I_B = 0$, $I_F = 0$	30			30			V
$V_{(BR)EBO}$ Emitter-Base Breakdown Voltage	$I_E = 10 \mu A$, $I_C = 0$, $I_F = 0$	7						V
$V_{(BR)ECO}$ Emitter-Collector Breakdown Voltage	$I_E = 10 \mu A$, $I_F = 0$				7			V
$I_{C(on)}$ On-State Collector Current	$V_{CE} = 1 V$, $I_B = 0$, $I_F = 10 mA$	30	100					mA
$I_{C(off)}$ Off-State Collector Current	$V_{CE} = 2 V$, $I_F = 10 mA$				30	160		nA
$I_{C(off)}$ Collector Current	$V_{CE} = 10 V$, $I_B = 0$, $I_F = 0$		100			100		nA
h_{FE} Transistor Static Forward Current Transfer Ratio	$V_{CE} = 1 V$, $I_C = 10 mA$, $I_F = 0$	15,000						
V_F Input Diode Static Forward Voltage	$I_F = 10 mA$		1.5			1.5		V
$V_{CE(sat)}$ Collector-Emitter Saturation Voltage	$I_C = 125 mA$, $I_B = 0$, $I_F = 50 mA$		1					V
r_{IO} Input-to-Output Internal Resistance	$I_C = 10 mA$, $I_F = 10 mA$					1		
r_{IO} Input-to-Output Internal Resistance	$V_{in-out} = \pm 1.5 kV$, See Note 6	10 ¹¹			10 ¹¹			Ω
C_{io} Input-to-Output Capacitance	$V_{in-out} = 0$, $f = 1 MHz$, See Note 6	1	1.3		1	1.3		pF

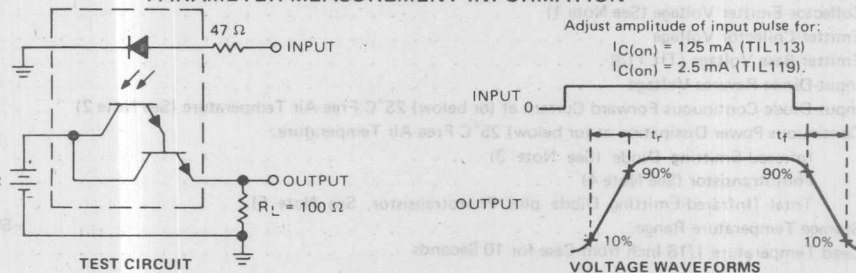
NOTE 6: These parameters are measured between both input-diode leads shorted together and all the phototransistor leads shorted together.

† References to the base are not applicable to the TIL119.

switching characteristics at 25°C free-air temperature

PARAMETER	TEST CONDITIONS	TIL113			TIL119			UNIT
		MIN	TYP	MAX	MIN	TYP	MAX	
t_r Rise Time	$V_{CC} = 15 V$, $I_{C(on)} = 125 mA$		50					μs
t_f Fall Time	$R_L = 100 \Omega$, See Figure 1		50					μs
t_r Rise Time	$V_{CC} = 10 V$, $I_{C(on)} = 2.5 mA$				50			μs
t_f Fall Time	$R_L = 100 \Omega$, See Figure 1				50			μs

PARAMETER MEASUREMENT INFORMATION



- NOTES: a. The input waveform is supplied by a generator with the following characteristics: $Z_{out} = 50 \Omega$, $t_r \leq 15 ns$, duty cycle $\approx 1\%$, $t_w = 100 \mu s$.
 b. The output waveform is monitored on an oscilloscope with the following characteristics: $t_r \leq 12 ns$, $R_{in} \geq 1 M\Omega$, $C_{in} \leq 20 pF$.

FIGURE 1—SWITCHING TIMES

TEXAS INSTRUMENTS
 INCORPORATED
 POST OFFICE BOX 5012 • DALLAS, TEXAS 75222

TYPES TIL113, TIL119 OPTO-COUPLEDERS

TYPICAL CHARACTERISTICS

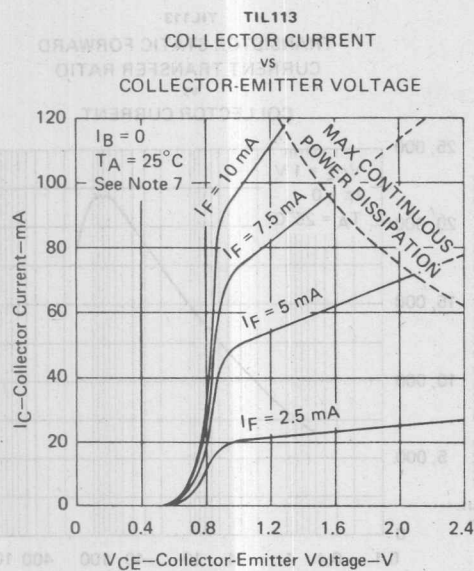


FIGURE 2

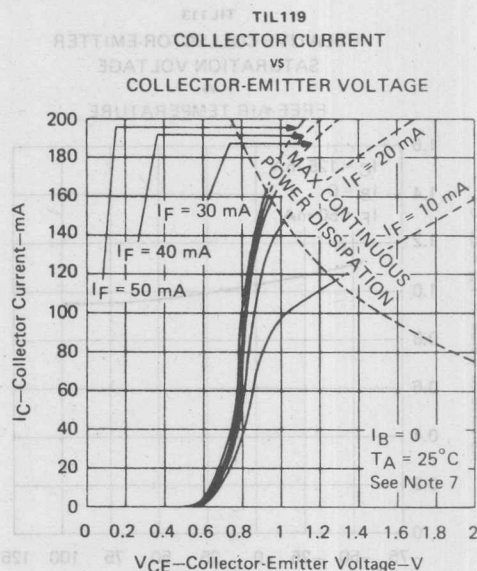


FIGURE 3

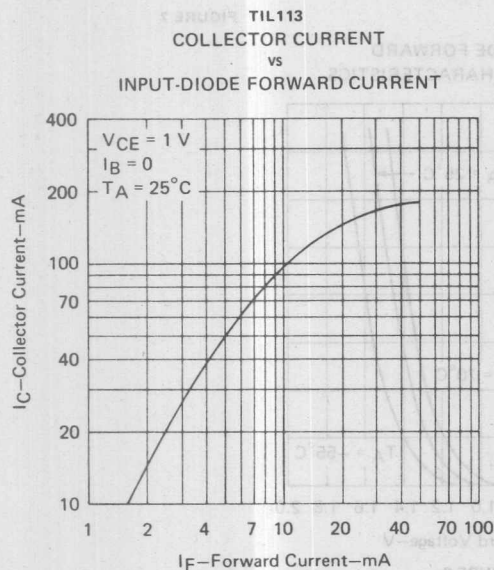


FIGURE 4

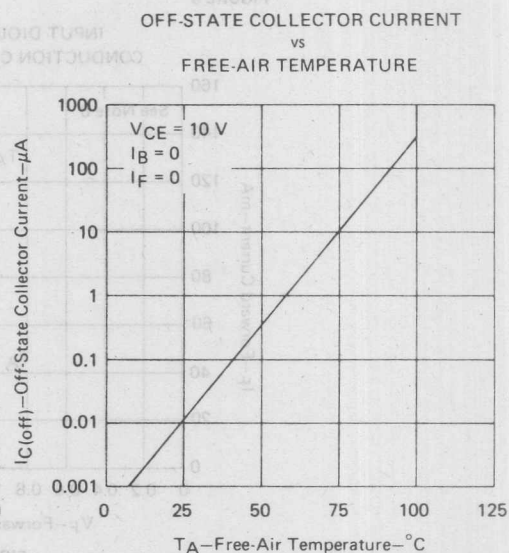


FIGURE 5

NOTE 7: Pulse operation of input diode is required for operation beyond limits shown by dotted line.

TEXAS INSTRUMENTS
INCORPORATED
POST OFFICE BOX 5012 • DALLAS, TEXAS 75222

TYPES TIL113, TIL119 OPTO-COUPLEDERS

TYPICAL CHARACTERISTICS

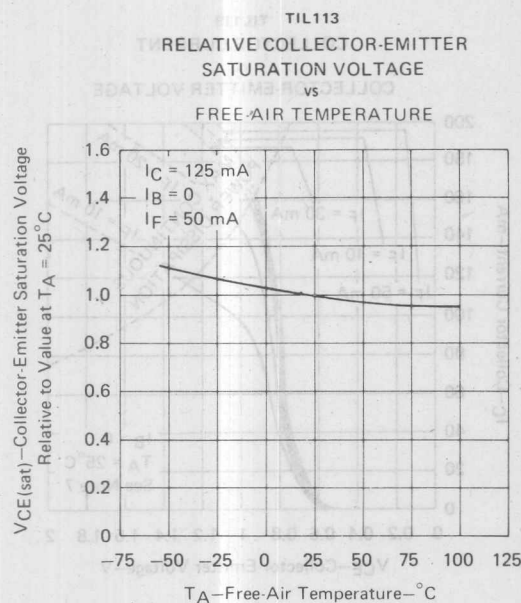


FIGURE 6

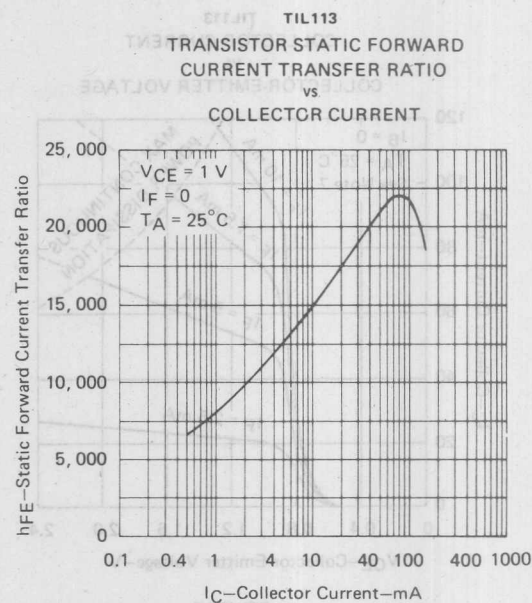


FIGURE 7

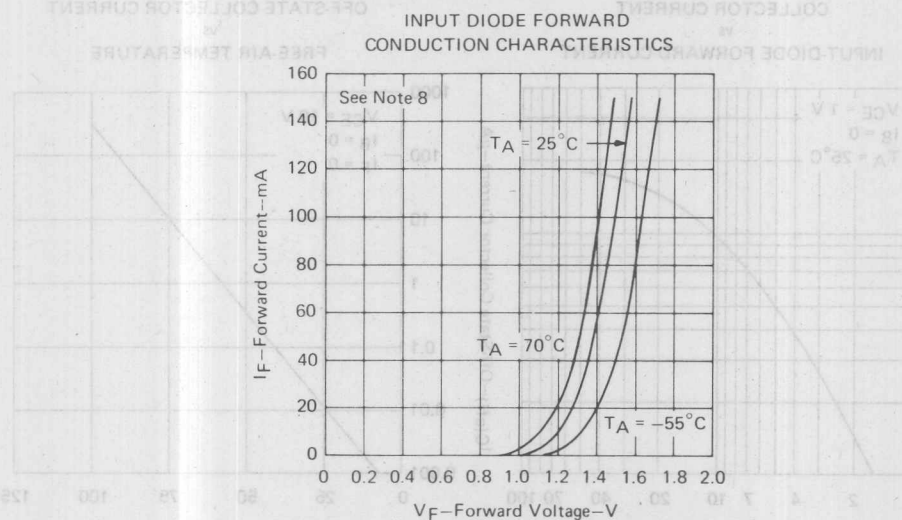


FIGURE 8

NOTE 8: This parameter was measured using pulse techniques. $t_w = 1\text{ ms}$, duty cycle $\leq 2\%$.

TEXAS INSTRUMENTS
INCORPORATED
POST OFFICE BOX 5012 • DALLAS, TEXAS 75222

PRINTED IN U.S.A.
TI cannot assume any responsibility for any circuits shown
or represent that they are free from patent infringement.

TEXAS INSTRUMENTS RESERVES THE RIGHT TO MAKE CHANGES AT ANY TIME
IN ORDER TO IMPROVE DESIGN AND TO SUPPLY THE BEST PRODUCT POSSIBLE

I/O Module Detail

Electrical Specifications

AC INPUT MODULES	MODEL IAC5	MODEL IAC15	MODEL IAC24	MODEL IAC5-A	MODEL IAC15-A	MODEL IAC24-A
AC INPUT LINE VOLTAGE	95 to 130 VAC			180 to 280 VAC		
INPUT CURRENT AT RATED LINE	10 ma					
ISOLATION INPUT TO OUTPUT	2500 Volt RMS					
INPUT ALLOWED FOR NO OUTPUT	1.5 ma					
TURN ON TIME	20 Millisecond Maximum					
TURN OFF TIME	20 Millisecond Maximum					
OUTPUT TRANST. BREAKDOWN	30 Volts DC					
OUTPUT CURRENT	25 ma					
OUTPUT LEAKAGE 30VDC, NO. INPUT	100 Microamp Maximum					
OUTPUT VOLTAGE DROP	.4 Volts at 25 ma Load					
LOGIC SUPPLY VOLTAGE DC	4.5 to 6 V	12 to 18 V	20 to 30 V	4.5 to 6 V	12 to 18 V	20 to 30 V
LOGIC SUPPLY CURRENT	12 ma	15 ma	18 ma	12 ma	15 ma	18 ma
AC OUTPUT MODULES	MODEL OAC5	MODEL OAC15	MODEL OAC24	MODEL OAC5-A	MODEL OAC15-A	MODEL OAC24-A
LINE VOLTAGE	12 to 140 VAC			24 to 280 VAC		
CURRENT RATING	3 Amps ^①					
1-CYCLE SURGE	55 Amps Peak					
SIGNAL INPUT RESISTANCE	220 Ohm	1K Ohm	2.2K Ohm	220 Ohm	1K Ohm	2.2K Ohm
SIGNAL PICKUP VOLTS DC	3V 8V Ald.*	9V 16V Ald.*	18V 32V Ald.*	3V 8V Ald.*	9V 16V Ald.*	18V 32V Ald.*
SIGNAL DROPOUT VOLTS DC	1 Volt					
PEAK REPETITIVE VOLTAGE	400V			500 Volts		
MAXIMUM CONTACT DROP	1.6V					
OFF STATE LEAKAGE	5 ma RMS					
MINIMUM LOAD CURRENT	20 ma					
ISOLATION INPUT TO OUTPUT	2500 Volts RMS					
CAPACITANCE INPUT TO OUTPUT	8 Pf					
STATIC DV/DT	200 Volts/Microsecond Min					
COMMUTATING DV/DT	Built in snubber (will commute .5.power factor loads)					

*Allowed

5842 Research Drive, Huntington Beach, California 92649 (714) 892-3313

Reprinted with permission from OPTO 22, March, 1979. All rights reserved.

DC INPUT MODULES	MODEL IDC5	MODEL IDC15	MODEL IDC24
INPUT LINE VOLTAGE	10-32 VDC		
INPUT CURRENT	32 ma at 32V		
ISOLATION INPUT TO OUTPUT	2500 Volt RMS		
CAPACITANCE INPUT TO OUTPUT	8 Pf		
INPUT ALLOWED FOR NO OUTPUT	2 ma		
TURN ON TIME	5 Millisecond Max		
TURN OFF TIME	5 Millisecond Max		
OUTPUT TRANST. BREAKDOWN	30 Volts DC		
OUTPUT CURRENT	25 ma		
OUTPUT LEAKAGE 30 VDC NO INPUT	100 Microamps Max		
OUTPUT VOLTAGE DROP	.4 Volt at 25 ma		
LOGIC SUPPLY VOLTAGE	4.5 to 6V	12 to 18V	20 to 30V
LOGIC SUPPLY CURRENT	12 ma	15 ma	18 ma
DC OUTPUT MODULES	MODEL ODC5	MODEL ODC15	MODEL ODC24
LOAD VOLTAGE RATING	60V DC		
OUTPUT CURRENT RATING	3 Amps ^①		
OFF STATE LEAKAGE	1 ma Max		
ISOLATION INPUT TO OUTPUT	2500 V RMS		
SIGNAL PICK UP VOLTAGE	3V 8V Ald.*	9V 18V Ald.*	18V 28V Ald.*
SIGNAL DROP OUT VOLTAGE	1Volt		
SIGNAL INPUT RESISTANCE	220 Ohm	1K Ohm	2.2K Ohm
1 SECOND SURGE	5 Amps		
TURN ON TIME	500 Microsecond		
TURN OFF TIME	2.5 Millisecond		

* Allowed

① Derate .033 Amps per degree C from 20° C

High Voltage DC Output Modules

DC OUTPUT MODULES	MODEL ODC5-A	MODEL ODC15-A	MODEL ODC24-A
LOAD VOLTAGE RATING	200V DC		
OUTPUT CURRENT RATING	1 Amps		
OFF STATE LEAKAGE	2 ma Max		
ISOLATION INPUT TO OUTPUT	2500 V RMS		
SIGNAL PICK UP VOLTAGE	3V 8V Aid.*	9V 18V Aid.*	18V 28V Aid.*
SIGNAL DROP OUT VOLTAGE	1Volt		
SIGNAL INPUT RESISTANCE	220 Ohm	1K Ohm	2.2K Ohm
1 SECOND SURGE	5 Amps		
TURN ON TIME	500 Microsecond		
TURN OFF TIME	2.5 Millisecond		

* Allowed

Fast Switching DC Input Modules

DC INPUT MODULES	MODEL IDC5-B	MODEL IDC15-B	MODEL IDC24-B
INPUT LINE VOLTAGE	4-16 VDC		
INPUT CURRENT	14 ma at 5V		
ISOLATION INPUT TO OUTPUT	2500 Volt RMS		
CAPACITANCE INPUT TO OUTPUT	8 Pf		
INPUT ALLOWED FOR NO OUTPUT	1 Volt		
TURN ON TIME	50 Microsecond Max		
TURN OFF TIME	100 Microsecond Max		
OUT TRANSISTOR BREAKDOWN	30 Volts DC		
OUTPUT CURRENT	25 ma		
OUTPUT LEAKAGE 30 VDC NO INPUT	100 Microamps Max		
OUTPUT VOLTAGE DROP	4 Volt at 25 ma		
LOGIC SUPPLY VOLTAGE	4.5 to 6V	12 to 18V	20 to 30V
LOGIC SUPPLY CURRENT	12 ma		

Data Sheet 778



APPENDIX C **PROGRAM SOURCE LISTINGS**

USING INTEL'S INDUSTRIAL CONTROL SERIES IN CONTROL APPLICATIONS

```

        $TITLE ('CONTROL TASK')
/*****
* This task handles the control and monitoring of
* four oven chambers.
*****/
1  CONTROLTASK$MODULE:
    Do;
2  1  DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
        MESSAGE$HEAD ADDRESS,
        MESSAGE$TAIL ADDRESS,
        TASK$HEAD ADDRESS,
        TASK$TAIL ADDRESS,
        EXCHANGE$LINK ADDRESS)';
3  1  DECLARE TRUE LITERALLY 'OFFH';
4  1  DECLARE FALSE LITERALLY 'COH';
5  1  DECLARE POOLEAN LITERALLY 'BYTE';
6  1  DECLARE FOREVER LITERALLY 'WHILE 1';
7  1  DECLARE MSG$HDR LITERALLY '
        LINK ADDRESS,
        LENGTH ADDRESS,
        TYPE BYTE,
        HOME$EX ADDRESS,
        RESP$EX ADDRESS';
8  1  DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE (
        MSG$HDR,
        REMAINDER(1) BYTE)';
        /* AIMSG.ELT - ANALOG INPUT REQUEST MESSAGE FORMAT */
9  1  DECLARE AIMSG LITERALLY 'STRUCTURE (
        MSG$HDR,
        STATUS ADDRESS,
        BASE$PTR ADDRESS,
        CHANNEL$GAIN ADDRESS,
        ARRAY$PTR ADDRESS,
        COUNT ADDRESS,
        ACTUAL$COUNT ADDRESS)';
        /* AITYP.ELT - ANALOG INPUT MESSAGE TYPES */
10 1  DECLARE AIREP LITERALLY '30',
        AISQS LITERALLY '31',
        AISQV LITERALLY '32',
        AIRAN LITERALLY '33';
11 1  Declare (n,k) byte;
12 1  Declare (MSG$PTR, LOCKOUT) address;
13 1  Declare (BLOCK0, BLOCK1, BLOCK2, BLOCK3) byte external;
14 1  Declare TOLERANCE(4) address external;
15 1  Declare TEMP(4) address external;
16 1  Declare SETPOINT(4) address external;
17 1  Declare T$AVERAGE(4) address;
18 1  Declare T$LAST(4) address;
19 1  Declare T$LAST$AVERAGE(4) address;
20 1  Declare T$t5(4) address;
21 1  Declare T$t10(4) address;
22 1  Declare STATUS(4) byte external;
23 1  Declare CRT$DISPLAY$LOCK(5) address external;

```

```

24 1 Declare TEMP$CALIBRATE(5) address external;
25 1 Declare DUMMY$EXCH(5) address external;
26 1 Declare TEMP$LOCKOUT$EXCH(5) address external;
27 1 Declare RQAIEX(5) address external;
28 1 Declare ASD$EXCH(5) address external;
29 1 Declare CONSTANT$LOCKOUT$EXCH(5) address external;
30 1 Declare CRT$STATUS$EXCH(5) address external;
31 1 Declare ALARM$MSG structure (MSG$HDR);
32 1 Declare CONVERT ai$msg;
/* This term is used to convey initial temperatures */
33 1 Declare CAL$TEMP based MSG$PTR structure (
    MSG$HDR,
    CAL address );
34 1 RQWAIT:
    Procedure (EXCH,MESSAGE) address external;
35 2 Declare (EXCH,MESSAGE) address;
36 2 end RQWAIT;
37 1 RQSEND:
    Procedure (EXCH,MESSAGE) external;
38 2 Declare (EXCH,MESSAGE) address;
39 2 end RQSEND;
40 1 RQACPT:
    Procedure (EXCH) address external;
41 2 Declare EXCH address;
42 2 end RQACPT;
43 1 Declare OVEN$INSTOL(4) byte data (
    01H,02H,04H,08H );
44 1 Declare OVEN$CAUTION(4) byte data (
    10H,20H,40H,80H );
45 1 Declare OVEN$DANGER(4) byte data (
    01H,02H,04H,08H );
46 1 Declare OVEN$ON$MASK(4) byte data (
    01H,02H,04H,08H );
47 1 Declare OVEN$HEATER(4) byte data (
    10H,20H,40H,80H );
48 1 Declare OVEN$RUN(4) byte data (
    10H,20H,40H,80H );
49 1 Declare OFFSET(4) address;
50 1 Declare TABLE(256) address data (
    200,201,202,203,204,205,206,207,208,209,
    209,210,211,212,213,214,215,216,217,218,
    219,220,221,222,223,224,225,226,227,228,
    229,230,231,232,233,235,236,237,238,239,
    240,241,243,244,245,247,248,249,250,251,
    252,254,256,257,258,259,260,261,263,265,
    266,267,268,269,270,271,273,274,276,278,
    279,280,282,284,285,287,288,289,290,291,
    293,295,296,298,299,300,302,304,305,307,
    309,309,310,312,314,315,318,320,322,324,
    326,328,330,332,334,336,338,340,342,344,
    346,348,350,352,354,356,358,360,362,364,
    366,368,370,372,374,376,378,380,382,385,
    388,390,392,395,398,400,402,405,407,410,
    412,415,418,420,423,426,428,430,433,436,
    439,441,444,447,451,454,457,460,463,466,

```



```

82 3      Do n=0 to 3;
84 4      T$AVERAGE(n)=(T$LAST(n)+TEMP(n))/2;
      /*Project temperatures into the future */
85 4      If T$AVERAGE(n)>=T$LAST$AVERAGE(n)
      then do;
87 5          T$t5(n)=(T$AVERAGE(n)-T$LAST$AVERAGE(n))*5)
              +T$LAST$AVERAGE(n);
88 5          T$t10(n)=(T$AVERAGE(n)-T$LAST$AVERAGE(n))*10)
              +T$LAST$AVERAGE(n);
89 5      end;
90 4      else do;
91 5          T$t5(n)=T$LAST$AVERAGE(n)-((T$LAST$AVERAGE(n)
              -T$AVERAGE(n))*5);
92 5          T$t10(n)=T$LAST$AVERAGE(n)-((T$LAST$AVERAGE(n)
              -T$AVERAGE(n))*10);
93 5      end;
      /* Update stored data */
94 4      T$LAST$AVERAGE(n)=T$AVERAGE(n);
95 4      T$LAST(n)=TEMP(n);
      /* Test for active oven */
96 4      MSC$PTR=RQWAIT (.CONSTANT$LOCKOUT$EXCH,0);
97 4      If ((BLOCK0 AND OVEN$ON$MASK(n))<>0)
      AND (TEMP(n)<>0)
      then do;
99 5          STATUS(n)=7;
100 5          BLOCK2=BLOCK2 OR OVEN$RUN(n);
      /* Test for an intolerance condition */
101 5          If SETPOINT(n)-TOLERANCE(n) < TEMP(n) AND
              SETPOINT(n)+TOLERANCE(n) > TEMP(n)
              then do;
103 6              STATUS(n)=7;
104 6              BLOCK1=BLOCK1 OR OVEN$IN$TOL(n);
105 6          end;
106 5          else BLOCK1=BLOCK1 AND NOT OVEN$IN$TOL(n);
      /* Test for a caution condition */
107 5          If SETPOINT(n)-TOLERANCE(n) > T$t5(n) OR
              SETPOINT(n)+TOLERANCE(n) < T$t5(n)
              then do;
109 6              STATUS(n)=14;
110 6              BLOCK1=BLOCK1 OR OVEN$CAUTION(n);
111 6          end;
112 5          else BLOCK1=BLOCK1 AND NOT OVEN$CAUTION(n);
      /* Test for a danger condition */
113 5          If SETPOINT(n)-TOLERANCE(n) > TEMP(n) OR
              SETPOINT(n)+TOLERANCE(n) < TEMP(n)
              then do;
115 6              STATUS(n)=21;
116 6              BLOCK2=BLOCK2 OR OVEN$DANGER(n);
117 6          end;
118 5          else BLOCK2=BLOCK2 AND NOT OVEN$DANGER(n);
      /* Handle control of heater elements */
119 5          If SETPOINT(n) > T$t10(n)
              then BLOCK3=BLOCK3 OR OVEN$HEATER(n);
              else BLOCK3=BLOCK3 AND NOT OVEN$HEATER(n);
121 5          end;
122 5      else do;
123 4          /* Turn everything off when operator shuts off oven */
124 5          BLOCK1=BLOCK1 AND NOT OVEN$IN$TOL(n);
125 5          BLOCK1=BLOCK1 AND NOT OVEN$CAUTION(n);
126 5          BLOCK3=BLOCK3 AND NOT OVEN$HEATER(n);

```



```

127 5      BLOCK2=BLOCK2 AND NOT OVEN$DANGER(n);
128 5      BLOCK2=BLOCK2 AND NOT OVEN$RUN(n);
129 5      STATUS(n)=F;
130 5      end;
131 4      Call RCSEND(.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
132 4      end;

/* Output data to real world */
133 3      OUTPUT(232)=BLOCK1;
134 3      OUTPUT(233)=BLOCK2;
135 3      OUTPUT(234)=BLOCK2;
136 3      end;
137 2      end CONTROL$TASK;
138 1      end CONTROL$TASK$MODULE;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0946H      2374D
VARIABLE AREA SIZE  = 0054H      84D
MAXIMUM STACK SIZE  = 0006H      6D
235 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

```

$TITLE('CRT PARAMETER TASK')
/*****
* This task is used to examine and update the *
* temperature setpoints and tolerances for *
* each of the four ovens. *
*****/
1 UPDATE$TASK:
Do;
$Include (:FF:COMMON.ELT)
2 1 = DECLARE TRUE LITERALLY 'OFFH';
3 1 = DECLARE FALSE LITERALLY 'OFFH';
4 1 = DECLARE BOOLEAN LITERALLY 'BYTE';
5 1 = DECLARE FOREVER LITERALLY 'WHILE 1';
$Include (:F0:MSGTYP.ELT)
6 1 = DECLARE DATATYPE LITERALLY '0',
= INT$TYPE LITERALLY '1',
= MISSED$INT$TYPE LITERALLY '2',
= TIMES$OUT$TYPE LITERALLY '3',
= FSSREQ$TYPE LITERALLY '4',
= UC$REQ$TYPE LITERALLY '5',
= FSSNAK$TYPE LITERALLY '6',
= CNTRL$C$TYPE LITERALLY '7',
= READ$TYPE LITERALLY '8',
= CLR$RD$TYPE LITERALLY '9',
= LAST$RD$TYPE LITERALLY '10',
= ALARM$TYPE LITERALLY '11',
= WRITE$TYPE LITERALLY '12';
$Include (:F0:MSG.ELT)

```

```

7 1 = DECLARE MSG$HDR LITERALLY '
    = LINK ADDRESS,
    = LENGTH ADDRESS,
    = TYPE BYTE,
    = HOME$EX ADDRESS,
    = RESP$EX ADDRESS';

2 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
    = MSG$HDR,
    = REMAINDER(1) BYTE)';
    $Include (:F0:THMSG.ELT)

9 1 = DECLARE TH$MSC LITERALLY 'STRUCTURE (
    = MSG$HDR,
    = STATUS ADDRESS,
    = BUFFER$ADR ADDRESS,
    = COUNT ADDRESS,
    = ACTUAL ADDRESS,
    = REMAINDER(128) BYTE)';

10 1 = DECLARE MIN$TH$MSC$LENGTH LITERALLY '17';
    $Include (:F0:CHAR.ELT)

    =
    = /* SPECIAL ASCII CHARACTERS */
    =

11 1 = DECLARE
    = NULL LITERALLY '0CH',
    = CONTROL$C LITERALLY '03H',
    = CONTROL$E LITERALLY '05H',
    = BELL LITERALLY '07H',
    = TAB LITERALLY '09H',
    = LF LITERALLY '0AH',
    = VT LITERALLY '0BH',
    = FF LITERALLY '0CH',
    = CR LITERALLY '0DH',
    = CONTROL$P LITERALLY '0EH',
    = CONTROL$Q LITERALLY '0FH',
    = CONTROL$R LITERALLY '10H',
    = CONTROL$S LITERALLY '11H',
    = CONTROL$X LITERALLY '12H',
    = CONTROL$Z LITERALLY '13H',
    = ESC LITERALLY '1BH',
    = QUOTE LITERALLY '22H',
    = LCA LITERALLY '51H',
    = LCZ LITERALLY '7AH',
    = RUBOUT LITERALLY '7FH',

    $Include (:F0:SYNCH.EXT)

12 1 = RQSEND:
    = PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;

13 2 = DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
    =

14 2 = END RQSEND;
    =

15 1 = RQWAIT:
    = PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;

16 2 = DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
    =

```

```

17 2 = END RQWAIT;
18 1 = RQACPT:
    = PROCEDURE (EXCHANGE$PTR) ADDRESS EXTERNAL;
19 2 = DECLARE EXCHANGE$PTR ADDRESS;
20 2 = END RQACPT;
21 1 = RQISND:
    = PROCEDURE (IED$PTR) EXTERNAL;
22 2 = DECLARE IED$PTR ADDRESS;
23 2 = END RQISND;
24 1 DECLARE TEMP$CALIBRATE(5) address external;
25 1 DECLARE UPDATE$EXCH(5) address external;
26 1 DECLARE CRT$STATUS$EXCH(5) address external;
27 1 DECLARE COMP$EXCH(5) address external;
28 1 DECLARE CONSTANT$LOCKOUT$EXCH(5) address external;
29 1 DECLARE RQOUTX(5) address external;
30 1 DECLARE RQINFX(5) address external;
31 1 DECLARE WORDS$EXCH(5) address external;
32 1 DECLARE SETPOINT(4) address external;
33 1 DECLARE TOLERANCE(4) address external;
34 1 DECLARE BUFFER2 address;
35 1 DECLARE MSG$PTR address;
36 1 DECLARE MSG structure (
    MSG$HDR,
    STATUS address,
    BUFFER$PTR address,
    COUNT address,
    ACTUAL address );
37 1 DECLARE CAL$TEMP structure (
    MSG$HDR,
    CAL address );
38 1 DECLARE UPD$MSG address;
39 1 DECLARE ENERGIZE based UPD$MSG structure (
    MSG$HDR,
    STATUS address,
    BUFFER$PTR address,
    COUNT address,
    ACTUAL address );
40 1 DECLARE ENABLE$MSG structure (
    MSG$HDR );
41 1 DECLARE BUFFER(80) byte;
42 1 DECLARE OVEN byte;
43 1 DEC$REP:
    Procedure (SOURCE,TARGET) external;
44 2 DECLARE (SOURCE,TARGET) address;
45 2 end DEC$REP;
46 1 ASC$2$BINARY:
    Procedure (SOURCE,TARGET,SIZE) byte external;
47 2 DECLARE (SOURCE,TARGET) address;
48 2 DECLARE SIZE byte;
49 2 end ASC$2$BINARY;
50 1 DECLARE MSGS1(20) byte data (
    ESC,'E','ENTER OVEN NUMBER-');

```

```

51 1 Declare MSG$2(28) byte data (
    CR,LF,
    'ENTER NEW SETPOINT-',
    'XXXX.X-' );
52 1 Declare MSG$3(29) byte data (
    CR,LF,
    'ENTER NEW TOLERANCE-',
    'XXXX.X-' );
53 1 Declare CALMSC(12) byte data (
    'TEMPERATURE-' );
54 1 Declare MSG$4(62) byte data (
    CR,LF,
    '(STATUS-(S), PARAMETERS-(P), CALIBRATE-(C))',
    CR,LF,
    'ENTER REQUEST-' );
55 1 Declare WAIT literally 'MSG$PTR=';
56 1 Declare FOR literally 'RQWAIT';
57 1 Declare START literally 'CALL';
58 1 Declare TASK literally 'RQSEND';
59 1 UPDATE:
    Procedure public;
    /* Initialize task at start-up time */
60 2 Do forever;
61 3 MSG.RESPSEX=.COMPSEXCH;
    /* Wait for request to enter task */
62 3 UPD$MSG=RQWAIT (.UPDATESEXCH,C);
    /* Get desired oven number from operator */
63 3 RQST$OVEN:
    MSG.BUFFER$PTR=.MSG$1;
64 3 MSG.TYPE=WRITE$TYPE;
65 3 MSG.COUNT=20;
66 3 Start task (.RQOUTX,.MSG);
67 3 Wait for (.COMPSEXCH,C);
    /* ... Input new number */
68 3 MSG.BUFFER$PTR=.BUFFER;
69 3 MSG.COUNT=255;
70 3 MSG.TYPE=CLR$RD$TYPE;
71 3 Start task (.RQINPX,.MSG);
72 3 Wait for (.COMPSEXCH,C);
73 3 OVEN=(BUFFER(0) AND 07H)-1;
74 3 If OVEN > 3 then go to RQST$OVEN;
    /* Display request and current setpoint */
76 3 GET$TEMP:
    Call move (28,.MSG$2,.BUFFER);
77 3 Call DEC$REP (.SETPOINT(oven),.BUFFER+21);
78 3 MSG.TYPE=WRITE$TYPE;
79 3 MSG.COUNT=28;
80 3 Start task (.RQOUTX,.MSG);
81 3 Wait for (.COMPSEXCH,C);
    /* ... Input new setpoint */
82 3 MSG.TYPE=CLR$RD$TYPE;
83 3 Start task (.RQINPX,.MSG);
84 3 Wait for (.COMPSEXCH,C);
85 3 If ASC$2$BINARY(.BUFFER,.BUFFER2,1)=0 OR BUFFER2 > 700
    then go to GET$TEMP;
87 3 If BUFFER2 <> 0
    then do;
89 4 Wait for (.CONSTANT$LOCKOUT$EXCH,C);
90 4 SETPOINT(oven)=BUFFER2;
91 4 Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);

```



```

92  4      end;
/* Display request and current tolerance */
93  3      GETSTOL:
          Call move (29,.MSG$3,.BUFFER);
          Call DEC$REP (.TOLERANCE(oven),.BUFFER+22);
94  3      MSG.TYPE=WRITE$TYPE;
95  3      MSG.COUNT=29;
96  3      Start task (.RQOUTX,.MSG);
97  3      Wait for (.COMP$EXCH,0);
98  3      /* ...Input new tolerance */
99  3      MSG.TYPE=CLR$RD$TYPE;
100 3      Start task (.RQINPX,.MSG);
101 3      Wait for (.COMP$EXCH,0);
102 3      If ASC$2$BINARY(.BUFFER,.BUFFER2,1)=0 OR BUFFER2 > 700
          then go$to GETSTOL;
104 3      If BUFFER2 <> 0
          then do;
105 4          Wait for (.CONSTANT$LOCKOUT$EXCH,0);
107 4          TOLERANCE(oven)=BUFFER2;
108 4          Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
109 4      end;
/* Ask operator if he is finished */
110 3      REQ$NEXT:
          MSG.TYPE=WRITE$TYPE;
          MSG.COUNT=62;
          MSG.BUFFER$PTR=.MSG$4;
          Start task (.RQOUTX,.MSG);
          Wait for (.COMP$EXCH,0);
114 3      /* ...Get his response */
          MSG.TYPE=CLR$RD$TYPE;
          MSG.BUFFER$PTR=.BUFFER;
          Start task (.RQINPX,.MSG);
          Wait for (.COMP$EXCH,0);
118 3      If (BUFFER(0) <> 'S' AND BUFFER(0) <> 'P'
          AND BUFFER(0) <> 'C')
          then go$to REQ$NEXT;
121 3      If BUFFER(0)='P'
          then go$to RQST$OVEN;
123 3      If BUFFER(0)='C'
          then do;
125 4          GET$CAL:
          MSG.TYPE=WRITE$TYPE;
          MSG.COUNT=12;
          MSG.BUFFER$PTR=.CALMSG;
          Start task (.RQOUTX,.MSG);
          Wait for (.COMP$EXCH,0);
129 4          MSG.TYPE=CLR$RD$TYPE;
130 4          MSG.BUFFER$PTR=.EUFER;
131 4          Start task (.RQINPX,.MSG);
132 4          Wait for (.COMP$EXCH,0);
133 4          If ASC$2$BINARY(.BUFFER,.BUFFER2,1) =0
          OR BUFFER2>250 OR BUFFER2<200
          then go$to GET$CAL;
136 4          CAL$TEMP.CAL=BUFFER2;
137 4          Call RQSEND (.TEMP$CALIBRATE,.CAL$TEMP);
138 4      end;

```

```

MODULE INFORMATION:
  CODE AREA SIZE = 03C3H 963D
  VARIABLE AREA SIZE = 007CH 124D
  MAXIMUM STACK SIZE = 0004H 4D
  264 LINES READ
  0 PROGRAM ERROR(S)
END OF PL/M-80 COMPILATION

139 3 ENERGEIZE.TYPE=100;
140 3 Start task (.CRT$STATUS$EXCH,UPD$MSG);

141 3 end;
142 2 end UPDATE;
143 1 end UPDATE$TASK;

$TITLE('CRT UPDATE TASK')
/*****
* This task is utilized to update the CRT ter-
* minal display with the current operating par-
* ameters. It will be entered upon sytem start-
* up, upon operator request, or when a problem
* exists with any of the activated ovens.
*****/

1 CRT$DATA$MODULE:
  Do;
  $INCLUDE (:PP:SYNCH.EXT)
2 1 = RQSEND:
  = PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;
3 2 = DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
  = END RQSEND;
4 2 =
  =
5 1 = RQWAIT:
  = PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
6 2 = DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
  =
  =
7 2 = END RQWAIT;
  =
8 1 = RQACPT:
  = PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
9 2 = DECLARE EXCHANGE$POINTER ADDRESS;
  =
10 2 = END RQACPT;
  =
11 1 = RQISND:
  = PROCEDURE (IED$PTR) EXTERNAL;
12 2 = DECLARE IED$PTR ADDRESS;
  =
  =
13 2 = END RQISND;
  = $INCLUDE (:PP:MSGTYP.ELT)
14 1 = DECLARE DATA$TYPE LITERALLY '0';

```

```

=          INT$TYPE LITERALLY '1',
=          MISSED$INT$TYPE LITERALLY '2',
=          TIMESOUT$TYPE LITERALLY '3',
=          FSSREQ$TYPE LITERALLY '4',
=          UCSREQ$TYPE LITERALLY '5',
=          FSSNAK$TYPE LITERALLY '6',
=          CNTRL$C$TYPE LITERALLY '7',
=          READ$TYPE LITERALLY '8',
=          CLR$RD$TYPE LITERALLY '9',
=          LAST$RD$TYPE LITERALLY '10',
=          ALARM$TYPE LITERALLY '11',
=          WRITE$TYPE LITERALLY '12';

$INCLUDE (:F0:EXCH.ELT)
15 1 = DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
=     MESSAGE$HEAD ADDRESS,
=     MESSAGE$TAIL ADDRESS,
=     TASK$HEAD ADDRESS,
=     TASK$TAIL ADDRESS,
=     EXCHANGE$LINK ADDRESS)';
$INCLUDE (:F0:COMMON.ELT)
16 1 = DECLARE TRUE LITERALLY 'OFFH';
17 1 = DECLARE FALSE LITERALLY 'OFFH';
18 1 = DECLARE BOOLEAN LITERALLY 'BYTE';
19 1 = DECLARE FOREVER LITERALLY 'WHILE 1';
$INCLUDE (:F0:MSG.ELT)
20 1 = DECLARE MSG$HDR LITERALLY '
=     LINK ADDRESS,
=     LENGTH ADDRESS,
=     TYPE BYTE,
=     HOME$EX ADDRESS,
=     RESP$EX ADDRESS';
21 1 = DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE (
=     MSG$HDR,
=     REMAINDER(1) BYTE)';
$INCLUDE (:F0:CHAR.ELT)
=     /* SPECIAL ASCII CHARACTERS */
=
22 1 = DECLARE
=     NULL          LITERALLY '00H',
=     CONTROL$C     LITERALLY '03H',
=     CONTROL$E     LITERALLY '05H',
=     BELL          LITERALLY '07H',
=     TAB           LITERALLY '09H',
=     LF            LITERALLY '0AH',
=     VT            LITERALLY '0BH',
=     FF            LITERALLY '0CH',
=     CR            LITERALLY '0DH',
=     CONTROL$P     LITERALLY '10H',
=     CONTROL$Q     LITERALLY '11H',
=     CONTROL$R     LITERALLY '12H',
=     CONTROL$S     LITERALLY '13H',
=     CONTROL$X     LITERALLY '18H',
=     CONTROL$Z     LITERALLY '1AH',
=     ESC           LITERALLY '1BH',
=     QUOTE         LITERALLY '22H',

```



```

44 1      Declare OVENS$ON(4) byte data (
        01H,02H,04H,08H );
45 1      Declare OVENS$CAUTION(4) byte data (
        10H,20H,40H,80H );
46 1      Declare CRT structure(
        MSG$HDR,
        STATUS address,
        BUFFER$PTR address,
        COUNT address,
        ACTUAL address );
47 1      Declare CRTLOCK structure (MSG$HDR);
48 1      Declare CRT$DISPLAY$LOCK(5) address external;
49 1      Declare TEMP$LOCKOUT$EXCH(5) address external;
50 1      Declare CONSTANT$LOCKOUT$EXCH(5) address external;
51 1      Declare CRT$EXCH(5) address external;
52 1      Declare CRT$STATUS$EXCH(5) address external;
53 1      Declare DUMMY$EXCH(5) address external;
54 1      Declare READ$BUFFER$EXCH(5) address external;
55 1      Declare UPDATE$EXCH(5) address external;
56 1      Declare RQINPX(5) address external;
57 1      Declare RQOUTX(5) address external;
58 1      Declare RQWAKE(5) address external;
59 1      Declare RQL7EX(5) address external;
60 1      Declare RQL6EX(5) address external;
61 1      Declare RQDBUG(5) address external;
62 1      Declare RQALRM(5) address external;
63 1      Declare TEMP(4) address external;
64 1      Declare DISP$TEMP(4) address;
65 1      Declare SETPOINT(4) address external;
66 1      Declare DISP$SETPNT(4) address;
67 1      Declare TOLERANCE(4) address external;
68 1      Declare DISP$TOL(4) address;
69 1      Declare STATUS(4) byte external;
70 1      Declare DISP$STAT(4) byte;
71 1      Declare (BLOCK1,BLOCK2) byte external;
72 1      Declare WORK$BUFF(170) byte;
73 1      Declare BUFFER$A(70) byte;
74 1      Declare (CHANGE,n,ALARM,NEW,BLANKER) byte;
75 1      Declare START literally 'call';
76 1      Declare TASK literally 'rqsend';
77 1      Declare WAIT literally 'msg$ptr=';
78 1      Declare For literally 'rqwait';
79 1      DEC$REP:
        Procedure(SOURCE,TARGET) external;
80 2      Declare (SOURCE,TARGET) address;
81 2      end DEC$REP;

```

```

82 1  CRT$DATA$TASK:
      Procedure public;
      /* Initialize system at start-up time */
83 2      Start task (.TEMP$LOCKOUT$EXCH,.STARTER(0));
84 2      Start task (.CONSTANT$LOCKOUT$EXCH,.STARTER(1));
85 2      STARTER(2).TYPE=100;
86 2      Start task (.CRT$STATUS$EXCH,.STARTER(2));
87 2      CRT.RESP$EX=.CRT$EXCH;
      /* Perform main CRT wait */
88 2      Do forever;
89 3          Wait for (.DUMMY$EXCH,10);
90 3          Wait for (.CRT$STATUS$EXCH,0);
91 3          If MSG.TYPE=255
          then ALARM=1;
          else ALARM=0;
93 3          /* Output heading */
          If (MSG.TYPE=100 OR MSG.TYPE=255)
          then do;
96 4              If ALARM=0
          then call RQSEND(.CRT$DISPLAY$LOCK,.CRT$LOCK);
98 4              CRT.TYPE=WRITE$TYPE;
99 4              CRT.COUNT=167;
100 4              CRT.BUFFER$PTR=.WORK$BUFF;
101 4              READ.TYPE=CLR$RD$TYPE;
102 4              READ.COUNT=255;
103 4              READ.RESP$EX=.READ$BUFFER$EXCH;
104 4              READ.BUFFER$PTR=.BUFFERA;
105 4              If ALARM=0
          then start task (.RQINPX,.READ);
          Call move (82,.CRT$HDR,.WORK$BUFF);
          Call move (86,.CRT$HDR+82,.WORK$BUFF+82);
107 4              Start task (.RQOUTX,.CRT);
108 4              Wait for (.CRT$EXCH,0);
109 4              NEW=1;
110 4              end;
111 4          /* Test for change in temperature of any oven */
112 4          CHANGE=0;
113 3          Wait for (.TEMP$LOCKOUT$EXCH,0);
114 3          Do n=0 to 3;
115 4              If TEMP(n)<>DISP$TEMP(n)
          then CHANGE=1;
116 4          end;
117 3          Call move (8,.TEMP,.DISP$TEMP);
118 3          Start task (.TEMP$LOCKOUT$EXCH,MSG$PTR);
119 3          /* When a change exists build new line */
120 3          If CHANGE OR NEW
          then do;
121 4              Call move (90,.L1$IMAGE,.WORK$BUFF);
122 4              Do n=0 to 3;
123 5                  Call DEC$REP(.DISP$TEMP(n),DISPLAY$PTR1(n));
124 5              end;
125 3          /* Output new temperature line to CRT */
126 3          CRT.TYPE=WRITE$TYPE;
127 3          CRT.COUNT=87;

```

```

129 4      Start task (.RQOUTX,.CRT);
130 4      Wait for (.CRT$EXCH,0);
131 4      end;
/* Test for change in oven setpoints */
132 3      CHANGE=0;
133 3      Wait for (.CONSTANT$LOCKOUT$EXCH,0);
134 3      Do n=0 to 3;
135 4          If SETPOINT(n)<>DISP$SETPNT(n)
              then CHANGE=1;
137 4      end;
138 3      Call move (8,.SETPOINT,.DISP$SETPNT);
139 3      Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
/* Build new line when a change was detected */
140 3      If CHANGE OR NEW
          then do;
142 4          Call move (92,.L2$IMAGE,.WORK$BUFF);
143 4          Do n=0 to 3;
144 5              Call DEC$REP(.DISP$SETPNT(n),DISPLAY$PTR2(n));
145 5          end;
/* Output setpoint line */
146 4      CRT.TYPE=WRITE$TYPE;
147 4      CRT.COUNT=89;
148 4      CRT.BUFFER$PTR=.WORK$BUFF;
149 4      Start task (.RQOUTX,.CRT);
150 4      Wait for (.CRT$EXCH,0);
151 4      end;
/* Test for change in tolerance line */
152 3      CHANGE=0;
153 3      Wait for (.CONSTANT$LOCKOUT$EXCH,0);
154 3      Do n=0 to 3;
155 4          If TOLERANCE(n)<>DISP$TOL(n)
              then CHANGE=1;
157 4      end;
158 3      Call move (8,.TOLERANCE,.DISP$TOL);
159 3      Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
/* When change is found, build new line */
160 3      If CHANGE OR NEW
          then do;
162 4          Call move (94,.L3$IMAGE,.WORK$BUFF);
163 4          Do n=0 to 3;
164 5              Call DEC$REP(.DISP$TOL(n),DISPLAY$PTR3(n));
165 5          end;
/* Output tolerance line */
166 4      CRT.TYPE=WRITE$TYPE;
167 4      CRT.COUNT=91;
168 4      CRT.BUFFER$PTR=.WORK$BUFF;
169 4      Start task (.RQOUTX,.CRT);
170 4      Wait for (.CRT$EXCH,0);
171 4      end;
/* Build status message */
172 3      CHANGE=0;
173 3      Wait for (.CONSTANT$LOCKOUT$EXCH,0);
174 3      Do n=0 to 3;
175 4          If STATUS(n)<>DISP$STAT(n)
              then CHANGE=1;

```



```

177 4      end;
178 3      Call move (4,.STATUS,.DISP$STAT);
179 3      Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
      /* Output to display */
180 3      If CHANGE OR NEW
      then do;
182 4          Call move (75,.L4IMAGE,.WORK$BUFF);
183 4          Do n=0 to 3;
184 5              Call move (7,.MESSAGES+DISP$STAT(n),DISPLAY$PTR4(
n));
185 5          end;
186 4          CRT.COUNT=76;
187 4          Start task (.RQOUTX,.CRT);
188 4          Wait for (.CRT$EXCH,0);
189 4      end;
      /* test for request to exit this mode */
190 3      MSG$PTR=RQACPT (.READ$BUFFER$EXCH);
191 3      If ALARM=0
      then do;
193 4          If (MSG$PTR <> 0 and EUPFERA(0) = 1FH)
      then do;
195 5              MSG$PTR=RQWAIT (.CRT$DISPLAY$LOCK,0);
196 5              start task (.UPDATE$EXCH,MSG$PTR);
197 5          end;
198 4          else do;
199 5              If MSG$PTR=0
      then STARTER(2).TYPE=200;
      else STARTER(2).TYPE=100;
201 5              Start task (.CRT$STATUS$EXCH,.STARTER(2));
202 5              NEW=0;
203 5          end;
204 5      end;
205 4      end;
206 3      end;
207 2      end CRT$DATA$TASK;
208 1      end CRT$DATA$MODULE;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0720H   1824D
VARIABLE AREA SIZE = 0189H   303D
MAXIMUM STACK SIZE = 0004H    4D
388 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

```

$TITLE('ASCII STRING TO FIXED BINARY')
/*****
* This program converts an ASCII string into a fixed point binary number. The fixed decimal point
* is determined by the parameter passed in SIZE.
*****/
1  ASC$2$BINARY$MODULE:
   Do;
   /* SPECIAL ASCII CHARACTERS */
2  1  DECLARE
      NULL          LITERALLY '00H';
      CONTROL$C     LITERALLY '03H';
      CONTROL$E     LITERALLY '05H';
      BELL          LITERALLY '07H';
      TAB          LITERALLY '09H';
      LF           LITERALLY '0AH';
      VT           LITERALLY '0BH';
      FF           LITERALLY '0CH';
      CR           LITERALLY '0DH';
      CONTROL$P     LITERALLY '10H';
      CONTROL$Q     LITERALLY '11H';
      CONTROL$R     LITERALLY '12H';
      CONTROL$S     LITERALLY '13H';
      CONTROL$X     LITERALLY '18H';
      CONTROL$Z     LITERALLY '1AH';
      ESC           LITERALLY '1BH';
      QUOTE         LITERALLY '22H';
      LCA           LITERALLY '61H';
      LCZ           LITERALLY '7AH';
      RUBOUT        LITERALLY '7FH';

3  1  ASC$2$BINARY:
      Procedure (SRC$PTR,TRGT$PTR,SIZE) byte public;
4  2      Declare (SRC$PTR,TRGT$PTR) address;
5  2      Declare (SOURCE based SRC$PTR) (3F) byte;
6  2      Declare RESULT based TRGT$PTR address;
7  2      Declare (N,SIZE,K,DP,DIGITS,VALID) byte;
8  2      Declare POWER(6) address data (
          0,1,10,100,1000,10000 );
      /* Find location of decimal point */
9  2      n=0;
10 2      Do while SOURCE(n)<>'.' AND SOURCE(n)<>CR
          AND SOURCE(n)<>LF;
11 3          n=n+1;
12 3      end;
13 2      DP=n;
      /* Provide correct number of digits to right of decimal */
14 2      Do n=0 to SIZE;
15 3          SOURCE(DP+n)=SOURCE(DP+n+1);
16 3          If SOURCE(DP+n)>3FH OR SOURCE(DP+n)<3FH
              then do k=n to SIZE;
18 4              SOURCE(DP+k)='0';
19 4          end;

```

```

20 3      end;
21 2      /* Mark end of string */
22 2      DIGITS=DP+SIZE;
23 2      /* Test for all valid characters */
24 3      VALID=1;
25 3      Do n=0 to DIGITS;
26 3          If SOURCE(n)>39H OR SOURCE(n)<30H
27 3              then VALID=0;
28 3      end;
29 2      If DIGITS>5
30 2          then VALID=0;
31 2      /* Convert data to binary and store */
32 2      n=0;
33 2      If VALID=1
34 2          then do;
35 3              RESULT=0;
36 3              Do while DIGITS>0;
37 4                  RESULT=RESULT+((
38 4                      SOURCE(n) AND 0FH) * POWER(DIGITS));
39 4                  n=n+1;
40 4                  DIGITS=DIGITS-1;
41 4              end;
42 3          end;
43 2      /* Return to calling program */
44 2      Return VALID;
45 2      end ASC$2$BINARY;
46 1      end ASC$2$BINARY$MODULE;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0178H 376D
VARIABLE AREA SIZE  = 000AH 10D
MAXIMUM STACK SIZE  = 0004H 4D
80 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

```

*****
STITLE('COMMON VARIABLE STORAGE')
/*****
* This module contains those variables common to *
* multiple tasks in the oven control application. *
*****/
1  VARIABLES$STORAGE:
Do;
2  1  Declare SETPOINT(4) address public;
3  1  Declare TOLERANCE(4) address public;
4  1  Declare TEMP(4) address public;
5  1  Declare STATUS(4) byte public;
6  1  Declare BLOCK0 byte public;
7  1  Declare BLOCK1 byte public;
8  1  Declare BLOCK2 byte public;
9  1  Declare BLOCK3 byte public;
10 1  end VARIABLE$STORAGE;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 0000F  0D
VARIABLE AREA SIZE  = 0020H  32D
MAXIMUM STACK SIZE  = 0000H  0D
16 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

```

MODULE INFORMATION:
CODE AREA SIZE      = 0000F  0D
VARIABLE AREA SIZE  = 0020H  32D
MAXIMUM STACK SIZE  = 0000H  0D
16 LINES READ
0 PROGRAM ERROR(S)
END OF PL/M-80 COMPILATION

```



```

STITLE('WORD TO ASCII CONVERSION')
/*****
* This routine converts a fixed point word in mem-
* ory into a 4 digit plus 1 decimal ASCII display-
* able number. Zero blanking is included.
*****/
1  DECSREP$MODULE:
Do;
2  1  DECSREP:
Procedure (SOURCE, TARGET) public;
3  2      Declare (SOURCE, TARGET) address;
4  2      Declare ANSWR(5) byte;
5  2      Declare (DISPLAY based TARGET)(5) byte;
6  2      Declare NUMBER based SOURCE structure (
          ELEMENT address);
7  2      Declare N byte;
8  2      Declare CALC(5) address;
/* Initialize */
9  2      Do n=0 to 4;
10 3          ANSWR(n)='0';
11 3      end;
12 2      CALC(0)=NUMBER.ELEMENT;
/* Convert to ASCII */
13 2      Do n=1 to 5;
14 3          CALC(n)=CALC(n-1)/10;
15 3          ANSWR(5-n)=(CALC(n-1) mod 10) + 30H;
16 3      end;
/* Perform zero blanking */
17 2      Do n=0 to 3;
18 3          If ANSWR(n)<>'0'
                then n=4;
20 3          else ANSWR(n)=' ';
21 3      end;
/* Format with decimal point */
22 2      Call move (4, .ANSWR, TARGET);
23 2      DISPLAY(4)='.';
24 2      DISPLAY(5)=ANSWR(4);
25 2      end DECSREP;
26 1  end DECSREP$MODULE;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 00EEH      238D
VARIABLE AREA SIZE  = 0014H      20D
MAXIMUM STACK SIZE  = 0004H      4D
40 LINES READ
0 PROGRAM ERROR(S)
END OF PL/M-80 COMPILATION

```

Closed Loop Control Using The iSBC 569/941™ Intelligent Digital Processors

Peter Andersen
OEM Manager

Andersen
OEM Microcomputer Systems Applications

multiple system bus. Processors and by minimizing traffic on the bus throughout by offloading the other bus master required. This technique greatly increases system interaction with other processes or devices is interrupts to the master or host processor is valuable processor time. These boards can save I/O can now be managed without testing.

Closed Loop Control The iSBC 560 Intelligent

Novemb

ing

041™

Get Digital

Processor **enter Andersen** **EM Microcomputer Systems**

November 1979

I. INTRODUCTION

The utilization of computers to provide control or monitoring functions for industrial processes frequently results in complex computer systems. Distributing the control and processing intelligence throughout the control network reduces significantly the complexity of the system while increasing the reliability. The physical areas being controlled or monitored by each portion of the distributed system will generally consist of a relatively small number of I/O functions which are related by some control algorithm.

The Intel iSBC 569 Intelligent Digital Controller (IDC) and the iSBC 941 Industrial Digital Processor (IDP) are a part of the expanding line of Intel products which are oriented toward filling the requirements of these systems. This application note deals with the use of these devices to provide control of a closed loop system using a version of the PID control algorithm.

It is assumed that the reader is familiar with the basic concepts required to generate software and has had some experience with using a computer. This application note will then guide the reader through a typical application, explaining in detail the decisions which must be made in order to effectively utilize a microcomputer to provide a control solution.

The application which has been chosen is considered to be typical of the type which lends itself to control. The mechanical aspects of the application will be explained so that the user not familiar with the particular machinery will be able to understand the development. It will be seen that the techniques used will apply to any other specific application.

The emphasis of the note will be on the use and implementation of the hardware and software features of the digital processor and controller. The actual PID control algorithm will not be developed in this application note.

Reasons for Intelligent Boards

The advent of microcomputers and the resulting trend toward utilizing these devices to control processes has resulted in many cases where the overall system performance has deteriorated because of the demands placed on the processor.

In these applications, the computer has become overburdened with control algorithms, alarm detection, communications, and the many other tasks required of it. The processor can be interrupted by time dependent tasks to the point where other processing tasks can not be completed.

Presently, Intel provides two I/O expansion boards which are capable of handling portions of the processing load which formally required processor time. These two devices are the iSBC 544 Intelligent Communications Controller and the iSBC 569 Intelligent Digital Controller. Tasks which involve communications or parallel digital I/O can now be offloaded without requiring valuable processor time. These boards can issue interrupts to the master or host processor if interaction with other processes or devices is required. This technique greatly increases system throughput by offloading the other bus master processors and by minimizing traffic on the Multibus system bus.

In some cases, it will be found that the intelligent controller can function to control the process in a stand-alone environment, providing a more functional, low cost control system.

The concept of offloading the processor of its input/output tasks can be developed on the iSBC 569 controller through the use of slave processors which may be installed on the board to assist the host. The result is the ability to provide up to four processors on a single intelligent slave I/O board by using the concept of slave processors.

The On-Board Slave Concept

The utilization of the iSBC 569 controller is enhanced through the use of On Board Slave processors (OBS). These devices distribute the system intelligence and offload the processor on the intelligent controller. They can provide custom digital interfaces with the various devices which may be connected to the I/O ports of the controller. The OBS device allows a designer to fully specify his control/interface algorithm in the peripheral chip without relying on the master processor. Three types of OBS compatible devices are available from Intel. These are: 1) Industrial Processors, 2) Standard UPI devices, and 3) UPI 8741A for custom applications. By combining the

devices in various combinations, optimum solutions can be generated for different control applications.

Before proceeding, we should cover the general characteristics of the OBS devices available for use in conjunction with the iSBC 569 controller. It will be seen that careful selection of the proper I/O controller chip can reduce significantly the design effort required to provide a control solution.

II. BASIC UNIVERSAL PERIPHERAL INTERFACE DISCUSSION

With the introduction of the Universal Peripheral Interface, Intel has expanded the intelligent peripheral concept by providing an intelligent controller that is fully user programmable. The 8741A is a complete single-chip microcomputer which connects directly to a master processor data bus.

To fully understand the techniques used by the UPI 8741A devices, we must have a general knowledge of their characteristics. Only then will we feel comfortable in implementing a design which uses the components.

Hardware Features

Each Universal Peripheral Interface has 1K bytes of program storage plus 64 bytes of RAM memory for data storage. It has a powerful, 8-bit CPU with a 2.5 μ sec cycle time and two interrupts. Over 90 instructions are provided in its instruction set. Most instructions are single byte and single cycle and none are more than two bytes long. These instructions are optimized for bit manipulation and I/O operations. Special instructions are included to allow binary or BCD arithmetic operations, table lookup routines, loop counters, and N-way branch routines.

The chip's 8-bit interval timer/event counter can be used to generate complex timing sequences for control applications or it can count external events such as switch closures and position encoder pulses. Software timing loops can be simplified or eliminated by the interval timer. If enabled, an interrupt to the CPU can occur when the timer overflows.

Two 8-bit bidirectional I/O ports are included which are TTL compatible. Each of the 16 port

lines can individually function as either input or output under software control.

The UPI microcomputer is fully supported with development tools. The combination of device features and Intel development support make the 8741A an ideal component for low-speed peripheral control applications.

Software Interface

The OBS communicates with the processor on the host board by means of data transfers between its registers and the host board's data bus. A communication protocol has been defined which provides a set of rules by which the processors may interact with each other. Two types of software protocol are currently defined. These are the "simple" and the "extended" protocol. Before attempting to utilize the OBS devices in an application, the concepts used for the communications must be fully understood.

When used on one of Intel's single board computers, the communication path is by means of the I/O ports on the host board. This means that two port addresses, an odd and an even location, are assigned to each OBS device. The even numbered port is used to transfer "data" between the processors. The odd numbered port is used to write commands into the OBS and to read its status. Each transfer between the host and the slave device consists of the movement of eight bits of information.

Four of the eight bits available in the status message have been given predefined functions. The bit will be set (logical 1) when the corresponding condition exists within the OBS device and will be reset (logical 0) when the condition does not exist. The functions of the four bits are:

Bit-0. Output Buffer Full (OBF).

This bit indicates that the OBS has placed information into the transfer register and that the information is available to the host processor. It can be read by performing an input operation from the even numbered port assigned to the particular OBS. When the data has been read, the bit will automatically be reset to indicate that no data is available. As we will see, this is one of the key features enabling efficient utilization of the host/

slave relationships on single board computers.

Bit-1. Input Buffer Full (IBF).

This bit is used to indicate that data has been placed into the input transfer register by the host device and that it has not yet been read by the slave. Data is transferred into the input register by means of the host performing an output to the even numbered port of the OBS. The bit will be reset when the device reads the data from the input transfer register into its accumulator. Data should only be output to the OBS when the IBF bit is reset!

Bit-2. F0 Flag.

Unlike the IBF and the OBF bits which are controlled by hardware, the F0 bit is controlled by the device software. The normal function of the flag is to provide a lockout to prevent the host from sending more data until the previous data has been processed or the operation is complete.

Bit-3. F1 is the Command/Data Flag.

It is automatically set when the host sends either a command (odd numbered port) or data (even numbered port). A logical 1 indicates that a command has been sent and a logical 0 indicates that data has been sent. This bit may also be cleared or toggled by the UPI software.

These bits will provide normal communications between the master and slave processors.

Figure 1 shows the sequence of operations which can be used by the host processor to establish communications with an OBS using the simple protocol. In Figure 1a, we see that all operations are initiated by the host. It will first verify that the IBF flag indicates that the input register is empty and available for receiving a command. The command is then sent to the odd numbered port. This command will inform the OBS that it is to perform some task. The task may involve a requirement for more information to be sent to the controller and it may involve the controller returning some data to the host. Figure 1b shows the operations required for receiving data from the OBS.

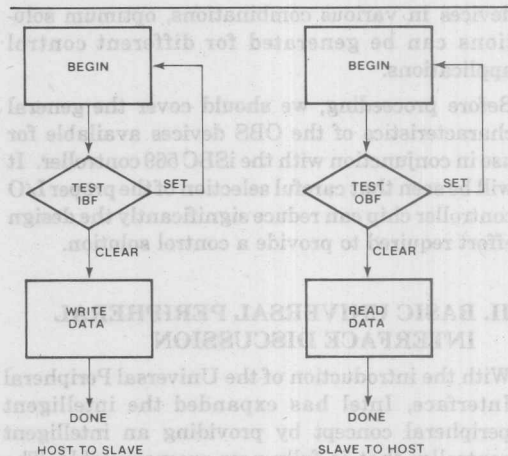


Figure 1. Simple Protocol

With these ideas in mind, we can move to a discussion of representative versions of the devices available for use on the IDC boards. We will then look at a typical application to see how they can actually be applied to solve a problem.

Standard Universal Peripheral Controllers

Intel presently manufactures three UPI controllers for non-industrial applications. These are:

1. 8278 Programmable Keyboard Interface
2. 8294 Data Encryption Unit
3. 8295 Dot Matrix Printer Controller

These devices offer an "off the shelf" solution to many applications which might be encountered.

The Intel 8278 is a general purpose programmable keyboard and display interface device. The keyboard portion can provide a scanned interface to 128-key contact or capacitive-coupled keyboards. The keys are fully debounced with N-key rollover and programmable error generation on multiple new key closures. Keyboard entries are stored in an 8-character FIFO with overrun status indication when more than 8-characters have been entered. Key entries set an interrupt request output to the master CPU. The display portion of the 8278 provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric displays and simple indicators may be used. The 8278 has a 16 x 4

display RAM which can be loaded or interrogated by the CPU. Both right entry calculator and left entry typewriter display formats are possible. Read and write of the display RAM can be done with auto-increment of the display RAM address.

The Intel 8294 Data Encryption Unit is designed to encode and decode 64-bit blocks of data using the algorithm specified in the Federal Information Processing Data Encryption Standard. The DEU operates on 64-bit test words using a 56-bit user specified key to produce 64-bit cipher words. The operation is reversible; if the cipher word is operated upon, the original test word is produced. Because the 8294 is compatible with the NBS encryption standard, it can be used in a variety of electronic funds transfer applications as well as other electronic banking and data handling applications where data must be encrypted.

Finally, the Intel 8295 Dot Matrix Printer Controller provides an interface to the LRC 7040 Series dot matrix impact printers. It may also be used as an interface to other similar printers. The chip may be used in a serial or parallel communication mode with the host processor. Furthermore, it provides internal buffering of up to 40 characters and contains a 7 x 7 matrix character generator which accommodates 64 ASCII characters.

Industrial Digital Processor

Intel produces the iSBC 941 Industrial Digital Processor (IDP) which is programmed to handle an assortment of typical industrial digital interfaces and transducers. The controller can function to provide any of the following:

1. Scan up to 16 inputs for a change of state.
2. Provide up to 8 gated one-shot outputs.
3. Provide eight gated outputs with programmable pulse widths and periods.
4. Provide monitoring of up to 8 input lines for event sensing or as a programmable divider.
5. Provide the period measurement of up to eight inputs.
6. Provide a frequency to count conversion of one input.
7. Provide for the control of a stepper motor having up to eight phases.
8. Provide a simplex asynchronous serial input.

9. Provide a simplex asynchronous serial output.

In addition to providing one of the above functions, the IDP can also handle simple parallel I/O through the unused port inputs or outputs.

III. FUNCTIONS OF THE INTELLIGENT DIGITAL CONTROLLER

The iSBC 569 Intelligent Digital Controller (IDC) is a versatile digital I/O processor. The IDC is designed to operate in a system using any one of the following three modes:

1. Intelligent Slave
2. Stand-alone System
3. Limited Bus Master

Additional power is obtained by the utilization of three OBS's to generate up to 48 parallel input/output data lines.

In the intelligent slave mode, the controller's RAM is shared between the on-board 8085A and the Multibus users via a dual-port controller. Thus, a single bus master can control several intelligent slaves using the dual-port RAM as the major communications path. Switches are provided on the board to allow the user to reserve 1K bytes of RAM for use by the 569's processor only. This reserved memory is not accessible via the Multibus system interface and does not occupy any bus address space.

In the stand-alone mode, the entire system can consist of a single IDC, with cables, power supply and enclosure. An IDC can be installed at a remote site as a completely autonomous system.

The IDC may also be operated as a limited bus master when it is the only bus master in the system. Expansion memory and I/O boards may be connected to the IDC via the Multibus system bus to increase the input/output capabilities. This mode could be used to configure one IDC as a bus master with additional IDC's as intelligent slaves. This mode is not available with any other bus masters such as iSBC single board computers, disk controllers, or DMA devices.

Input/Output Functions

The I/O interface between the iSBC 569 Intelligent Digital Controller and the external devices to

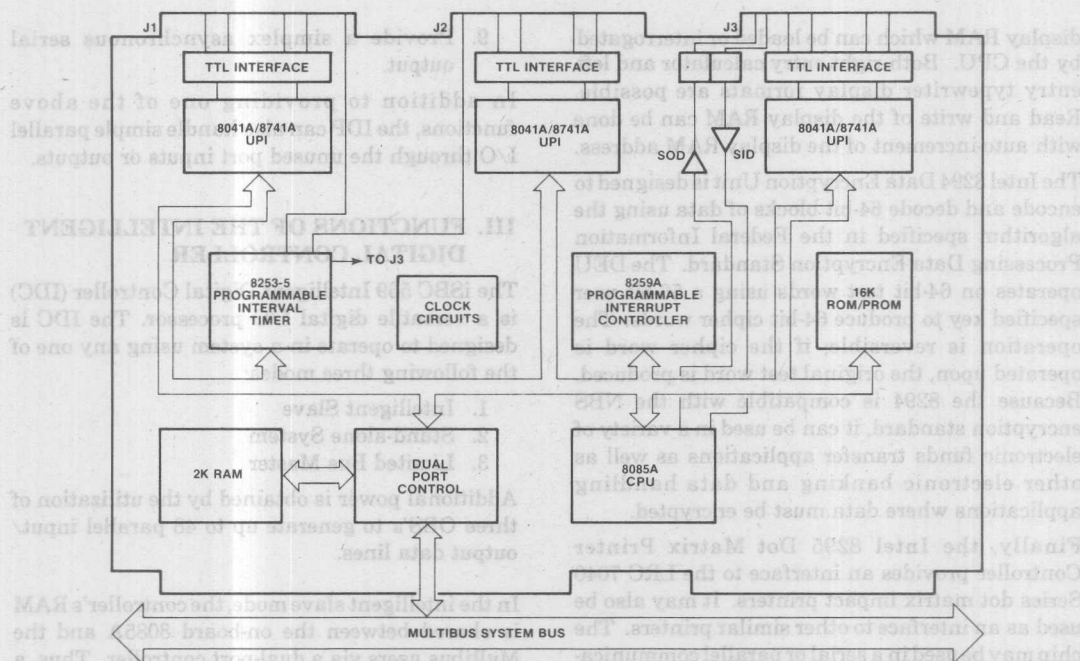


Figure 2. IDC Functional Block Diagram

which it is to be connected normally consists of various OBS devices. Each of these slaves has the ability to provide sixteen individual input and/or output lines. In addition, each provides two specialized input lines. The IDC is designed to accommodate up to three slave devices, so the normal I/O configuration of the board will consist of 48 digital data lines. If the specialized lines are considered, this number could be raised to 54. Sockets are provided for the insertion of drivers or terminators for use on the 48 digital lines. The 6 special purpose lines can only be used as inputs and are provided with pull-up resistors to terminate the input signals.

The driver/termination socket configuration limits the grouping of the I/O lines to be in groups of four. Any slave data line being used for an input must have its output latch placed into a logical 1 state so as to allow the input line to be controlled by the external signal.

IV. APPLICATION EXAMPLE

An example of the iSBC 569 controller in an application will help to explain the techniques used to implement a control system and to interface between the various functional units. The application chosen will consist of a typical use but will be simple enough to allow the design operations to be easily followed.

Suppose we choose to design a control system which will be produced as a subsystem to interface with and control a liquid applicator. As we go through the steps required to design and implement such a control system, we will see how the various hardware and software tools which are available from Intel can be utilized to allow easy implementation of the task.

Before proceeding, we will spend some time to insure there is a clear understanding about the definition of the liquid applicator. When this definition is complete, the design of the control subsystem can begin.

A liquid applicator consists of two functional parts: a device to control the flow of a solid material, and a device to control the flow of a liquid onto the material. We will actually be controlling two continuous process loops which are related by an input parameter which specifies the percentage of liquid to be applied to the dry material.

Figure 3 shows the components making up a typical weighbelt feeder. The operation of the feeder is straightforward. The vertical gate is adjusted manually to provide a desired gap between the conveyor belt and the lower portion of the gate. This will result in a nearly level distribution of material on the belt when it is moving. The weighbelt is connected to a load cell to provide information back to the control system giving the amount of weight on the belt at any instant. If we know the speed of the conveyor, it is simple to compute the amount of material flowing through the feeder during any time period. This

flow rate is known as the Mass Flow and is usually expressed as pounds per minute. The control of the feeder system can be provided by varying the belt speed until the desired flow rate has been obtained.

Our control system will be designed to control the belt speed and to monitor the weighbelt weight and any other parameters which we determine will be necessary to control the flow of material. A typical control process will require an optimum flow rate be established for each material of a different density. With a known material flow through the feeder, we can go about the process of applying the liquid flow to the material in order to complete our application example.

The second loop of the example will involve adding the liquid to the material coming from the feeder mechanism described above. Normally, the percentage of material to be applied is fixed by the

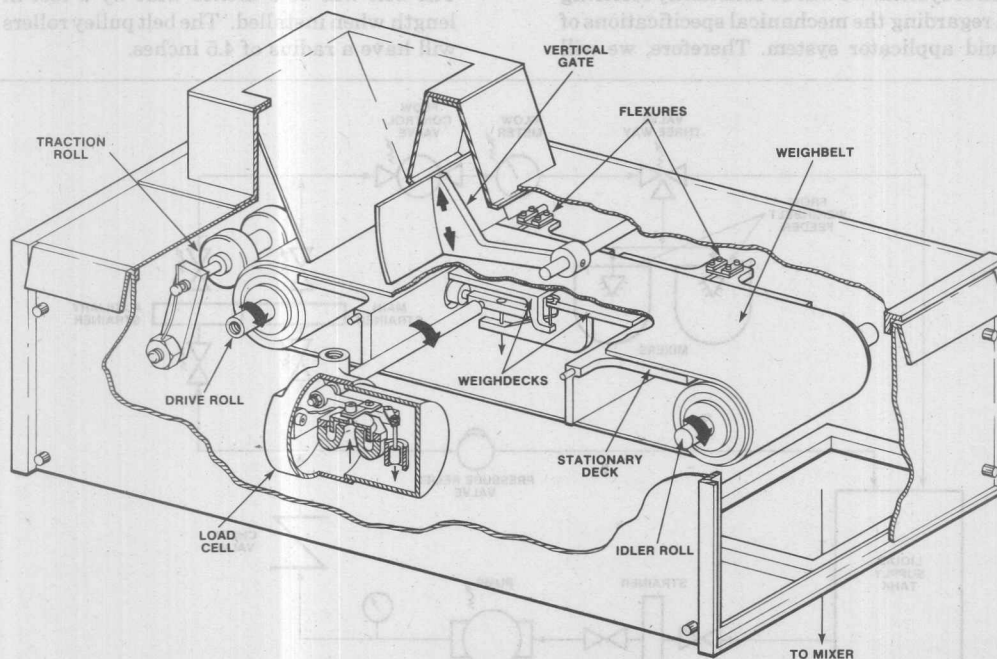


Figure 3. A Weighbelt Feeder

formula or mix design of the product which we are manufacturing. However, since the flow rate through the weighbelt feeder can and does vary (our first control loop will not always be able to exactly control the flow due to many conditions beyond our control), the liquid setpoint will constantly be changing as a function of the actual mass flow and the liquid percentage.

Figure 4 shows the liquid application piping diagram for the liquid portion of the control system. The items with which we will be directly concerned are the liquid flow meter and the control valve. The other components, while requiring consideration in an actual implementation, will be ignored in this application note for the sake of clarity. Let us consider the details of each control loop in more depth before we attempt to design the control system.

Mechanical Specifications

In subsequent portions involving development of the control system, we will be constantly referring to data regarding the mechanical specifications of the liquid applicator system. Therefore, we will

establish a set of theoretical technical specifications for our system. Later, we will see how close the control system can come to providing a control which meets or exceeds these parameters. These specifications will be broken down into two sets of data, one for physical parameters over which we have no control, and a second for the desired control characteristics.

The physical data provides information on the mechanical design and will be used for guidelines in selecting interface equipment and in preparing software algorithms. The physical data is:

Operating Belt Speed —

1.1 to 180 feet per minute. Adjusted by a variable speed motor directly coupled to the belt pulley mechanism.

Feed Output Rates —

Adjustable over a 10:1 range with a maximum output of 960 pounds per minute.

Feeder Belt Characteristics —

The belt will be 9 inches wide by 2 feet in length when installed. The belt pulley rollers will have a radius of 4.5 inches.

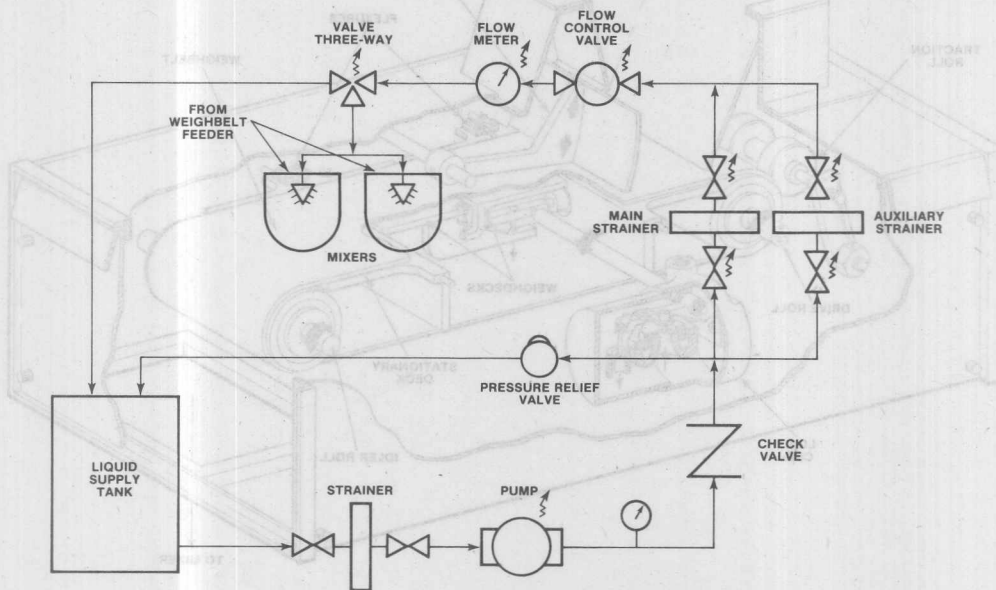


Figure 4. Liquid Flow Diagram

Feeder Weight Sensor —

The weighbelt feeder will incorporate a strain gauge load cell to measure the weight on the belt. Its linearity shall provide 0.1% of full scale range.

Liquid Flow Rates —

The liquid flow rates shall vary between 10.0 and 120.0 pounds per minute.

The desired operating characteristics of our control system will provide the following general responses:

Feeder Accuracy —

1% of full scale over a 10:1 range. The feeder will maintain the set feed rate within 1% of full scale over any one minute period. The minimum sample must be at least one pound.

Liquid Accuracy —

1% of full scale over the operating range. Must be able to track mass flow variations within the above limits.

These specifications will provide guidelines for the decisions which we will later make in providing a micro-computer control solution to the weighbelt feeder application.

Interface Requirements

A logical place to begin the consideration of the control system design is to examine the interface requirements and define the characteristics of the interfaces which will be required to implement the control. We will consider each element of the physical system separately.

Weighbelt Weight

The weighbelt weight will be sensed using a lever system connected to a load cell integral to the mechanical unit. The output of a strain gauge load cell is a low level (approximately 20 millivolts at full scale) analog output. Obviously, this signal must be somehow converted into a digital level before we can use its information to compute the actual mass flow across our weighbelt feeder. Our design process must define the characteristics of the digital signal so that the appropriate analog to digital converter system can be chosen. The design path can take any of several equally valid approaches, any of which will provide a functional control system. For the purposes of this

application note, we will assume that the design path will utilize the Intel iSBC 569 Intelligent Digital Processor.

This assumption requires us to utilize only signals which can be generated or interpreted using the computer board and its associated OBS's. We will not be capable of handling an analog signal. Since some type of signal conditioning would be required of the low level analog voltage anyway, this does not impose any serious restrictions on our design. Indeed, it will cause us to consider a technique which provides excellent noise rejection characteristics. We will assume that a voltage to frequency converter (V/F) will be installed near the load cell and the frequency will then be transmitted over a pair of wires to our digital interface. Commercially available converters provide a frequency output which varies between 0 and 10 kilohertz. With this in mind, we can continue with the development of the interfaces required in the application.

The load cell transducer will incorporate a local unit which generates a pulse train whose frequency is proportional to the weight upon the load cell. This mechanical arrangement is typical of many gravimetric feeder systems in use today.

For purposes of this application, it will be assumed that the system will be calibrated such that a weight of 10.00 pounds on the weighbelt will produce a pulse train frequency of 10 khz. No weight on the belt will generate a frequency of less than 30 hertz. The accuracy of the pulse output will be guaranteed to be proportional to the weight within 0.05%. Again, this is typical of devices available and in general use in similar applications.

The characteristics we have described above fall within the performance range of the iSBC 941 processor when operated in its frequency to count mode. If we assume a sample rate of 200 msec (this value should provide an adequate control characteristic since it is unlikely that the mechanical equipment can respond rapidly enough to warrant a faster control and sample time), the frequency count read by the iSBC 941 counter will range between 6 and 2000. System accuracy of reading the belt weight will thus exceed 0.1% of the full scale weight reading.

We will discuss the electrical and programming interfaces in subsequent sections of the application note.

Weighbelt Motor Control

The flow on the weighbelt will be controlled by changing the speed of the belt movement. Since the weighbelt is mechanically designed to maintain a constant bed level, the amount of material flowing will thus be adjusted.

The belt speed has traditionally been adjusted using either SCR controllers or by using variable transmissions between the motor and the conveyor belt. The increased utilization and development of stepper motors is leading toward greater use of direct stepper motor drives. This is the mode which will be utilized for this application.

The manufacturer's specifications for the weighbelt indicate that the following requirements exist for driving the device:

REQUIRED TORQUE — 149 LB-IN-IN

REQUIRED MAX SPEED — 2.54 REV/SEC.

Referring to typical manufacturer specification sheets for stepper motors, we find the torque vs. speed characteristics shown in Figure 5. Our application requires 2.54 revolutions/sec which translates to 508 steps per second when the stepper is used in a 1.8 degree per step mode. We can see that the requirements fall well within the capabilities of the particular motor.

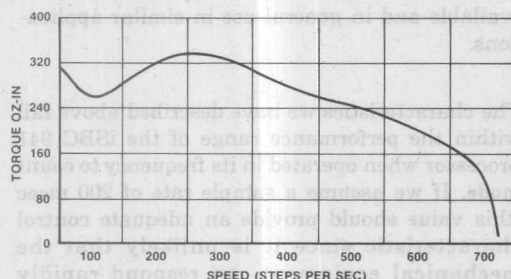


Figure 5. Stepper Motor Torque/Speed

At this point, we have four routes which may be pursued to actually interface with the motor. These are:

1. Utilize the iSBC 941 stepper mode to drive the stepper motor directly.
2. Utilize the iSBC 941 frequency generation mode to drive a standard stepper translator.
3. Utilize parallel outputs to provide a digital output to a stepper translator.
4. Utilize a 4-20 ma. current signal to a stepper translator.

Three of the above modes use a translator to drive the motor. If possible, we should strive to eliminate the cost of this intermediate device.

Again, we will refer to the published motor specification sheets. For our typical motor, the data is shown in Figure 6. The requirement for providing in excess of six amperes per winding exceeds the capabilities of the output drivers which can be installed on the iCS 930 termination board. We will be forced to either design a custom high power driver board or to use a translator module. To keep the application as simple as possible, we will choose the latter.

ELECTRICAL RATINGS 1.8 DEGREE STEPPING MOTOR

Motor Type	Time for One Step	DC Volts	Amperes Per Winding	Resistance Ohms	Inductance Millihenries
Ourtype	1.7 msec	2.3	6.1	0.37	2.4

Figure 6. Stepper Electrical Ratings

We have three choices left when the decision has been made to use a translator module. The use of a current output mode will necessitate the use of an external analog board. This is undesirable, both from the standpoint of interboard communication requirements, and from a cost effective basis.

The use of a parallel output would commit many of our output data ports and would require the installation of UPI modules or iSBC 941 modules to get the parallel output drivers. In addition, parallel digital input is not a common option of commercially available translators.

This leaves us with the use of a variable frequency output to provide stepping information to the translator module. This is a normal operational mode of the iSBC 941 processor and the required 508 hertz is within the normal output range of the device.

A definite advantage of our decision to use a stepper motor drive for the weighbelt is that we do not have to maintain accurate feedback and control algorithms to maintain the conveyor speed. Only a simple check need be made to verify that the conveyor has not stalled. The stepper motor will inherently maintain a speed proportional to the frequency rate.

The actual electrical and programming interfaces will be discussed in subsequent sections of this application note.

Weighbelt Speed Measurement

We have mentioned that a control system using a stepper motor for speed control can operate effectively in an open loop configuration. However, since a faulty component could result in failure of the motor to run, we must verify that the belt is indeed moving. This is easily accomplished by adding a magnetic sensor to the weighbelt rollers and counting the pulses generated as the device operates.

Typical magnetic sensors and ring magnets for installation on the weighbelt will provide us with ten pulses per revolution of a belt pulley. Since the pulley is operating at a maximum speed of 2.54 revolutions per second, we will receive between 0 and 25.4 counts per second. Using our sample period of 200 milliseconds, this means that we will count between 0 and 5 counts during each time interval. Our decision to use a stepper control loop rather than a conventional closed loop seems justified as we would obtain rather poor control with feedback having this poor of resolution.

We must make a decision to determine how the speed will be sensed by the control board. An obvious choice would be the use of an iSBC 941 processor operating in the period measurement mode. This would require using our third socket on the iSBC 569 host board and would leave us without the ability to use an additional device to support the liquid control loop. We should seek an alternative solution.

The iSBC 569 controller board provides an 8253 programmable interval timer. A first approach might be to attempt to configure one of these counters to provide an event counting mode and read the belt speed from the counter. However, this is not possible since we would be required to zero the counter after each reading and the counter does not load the preset count until a clock pulse is present. We would have no ability to distinguish between no belt motion and the belt motion which is the same as the previous reading!

An alternative approach is to create a software counter by routing the belt movement pulse to one of our interrupts and creating a program which will increment a counter. Each time a count is sensed, the software will increment a memory location by an increment which corresponds to the speed represented by one count.

Again, we will delay the discussion of the electrical and programming interfaces until subsequent sections of this application note.

Liquid Flow Control

The design of a control system to provide control of flow through a liquid valve is an integral part of the liquid pipe and plumbing design. To optimize the system operation and provide a system at the minimum cost, the integration of control and mechanical design must be made.

Several possibilities exist when making a decision as to which control valve to use in adjusting the liquid flow rate. The actual selection of the physical valve mechanism should be based upon the characteristics of the liquid flow. This decision is outside of the scope of this application note and will not be pursued. However, the valve actuator is a device which becomes an integral part of the control system and its selection is a function of the control system design.

Figure 7 shows the common control valve types which are used to vary the flow rate of liquids. The automatic control system we are designing precludes the use of a manual valve, so we must make our selection between the air actuated and the motorized control valve.

Classical control design has utilized air actuated valves almost exclusively. This type of actuator incorporates an intermediate transducer to

PROPORTIONAL CONTROL VALVES

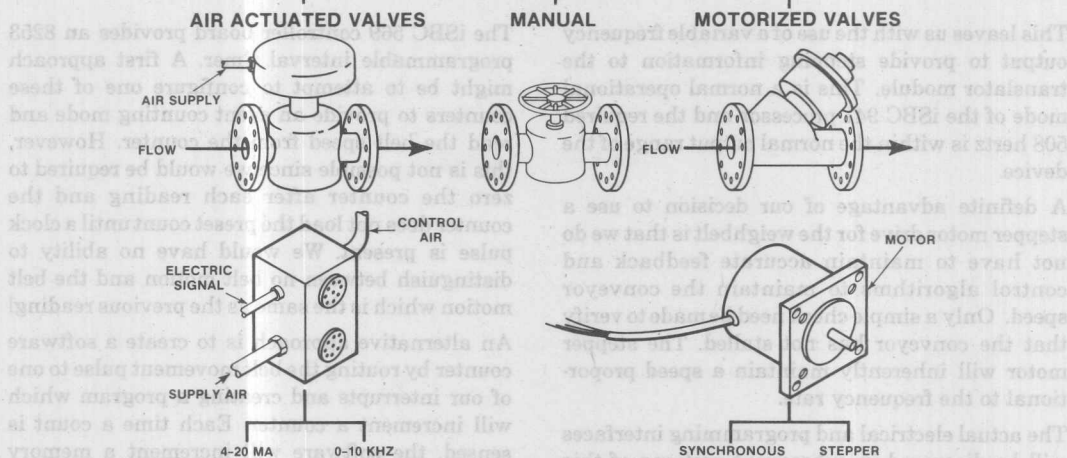


Figure 7. Control Valve Family

convert the signal generated by the control system into a variable air pressure. This air is used to drive a pneumatic control actuator. Two types of electrical to pneumatic transducers are in common use. The most prevalent converts a 4 to 20 milliamperes control signal into a proportional air signal. The second type will accept a 0 to 10 kHz pulse train and convert this to an air output.

Both of the above systems provide excellent electrical noise immunity and give reliable operation in industrial environments. They do, however, have disadvantages. A supply of air must be present at the control devices and this air must be maintained such that it is free from water and oil. In many cases, this presents costly installation and maintenance considerations. The use of computerized control systems has led to a recent concept of eliminating the intermediate conversion and using instead a digitally controlled actuator.

A stepper motor can be connected to the actuator

of the control valve to provide a simple and economical control path. The control outputs from the PID control loop can be sent to the iSBC 941 processor's command queue and the controller will handle the motor movements.

The electrical and programming interfaces of this interface will be fully discussed in subsequent sections.

Liquid Flow Measurement

The use of a liquid control valve to vary the liquid flow cannot in itself provide an accurate control loop. Because the flow rate through a fixed valve will vary with material densities, temperatures, and pressures, we must provide some type of feedback into our control algorithm. Thus, a flowmeter must be inserted into the liquid flow and its output returned to the system.

The control system designer can choose from several types of flow meters depending upon his requirements. Figure 8 shows many of the more

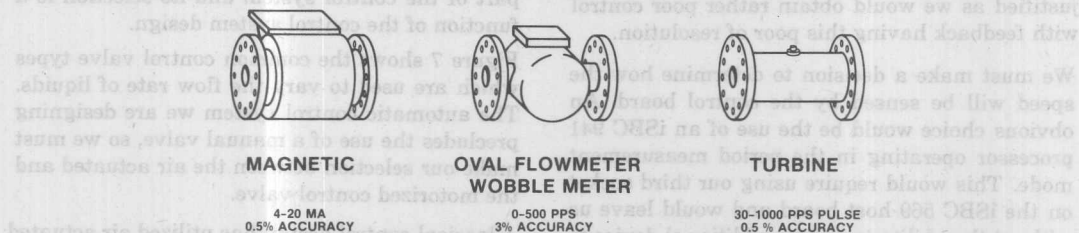


Figure 8. Flow Meter Classifications

standard classifications of flow meters. Our selection of the meter must take into account the type of electrical interface available from the meters. In our attempt to maintain a digital system which does not require additional support boards, we will reject the use of a magnetic flowmeter because this type of meter provides an analog type of output which would require the addition of another board into our control system. The wobble meter provides a digital pulse type output but its accuracy tends to discourage its use in a refined control loop. We will utilize the turbine meter for our liquid flow application.

The output of a turbine meter is a low voltage, low current AC signal whose frequency is proportional to the liquid flow rate. The manufacturers of the meters provide pre-amplifiers which convert the signal into 10 volt peak to peak square waves which are equivalent in frequency to the AC pulses. The operating frequency ranges typically from 100 to 1200 pulses per second.

It is desirable to measure the flow rate using a single iSBC 569 controller. If we consider that a 200 millisecond control interval will be used, the flow will result in a reading of between 20 and 240 pulses per sample period. These readings could be performed using an iSBC 941 processor, but we do not have the socket available for a fourth module, so we must consider utilizing another interrupt driven software counter as was done with the belt speed.

All control and monitoring equipment for our liquid control application has now been defined in such a manner as to be compatible with the utilization of a single iSBC 569 controller board. The actual interfaces to perform the interconnections and to provide control instructions can soon be considered.

Operator Interface

Finally, we must define the data communications which must take place between the controller, other system tasks, and the operator. Let us first consider the system control variables and the data which, if generated by the control process, might be useful to the remainder of the control system.

The first variable which comes to mind is the liquid flow setpoint. If we consider the entire

control system, this parameter will be found to be actually expressed as a percentage of the total output material. For example, if we assume the recipe required the final product to consist of 5% liquid by weight, we would require that our control system add the correct amount of liquid to perform this task.

To allow maximum flexibility of the control system, we should allow selection of various density materials onto the weighbelt. A host processor with computational capabilities can calculate the optimum gravimetric feeder flow rate for the materials being combined.

The control system can provide an integration function to allow totalization of the amount of material which has been transferred through the system. A capability of outputting the amount of material which has passed over the weighbelt and the amount of liquid added will be included.

The implications of the parameter storage and generation will be dealt with later when the host/slave relationships of the iSBC 569 controller are discussed.

Interface Summary

We have defined the required interfaces which will be needed to perform our control task. These can be grouped into external and internal interfaces. The external interfaces are those which connect to physical pieces of external equipment.

These are summarized in Figure 9. The internal interface relates to the data which is to be passed between the iSBC 569 Intelligent Slave board and other boards which may be present on the MULTIBUS system bus. These data areas are shown in Figure 10.

V. HARDWARE CONFIGURATION

We have now defined the various components which we will utilize on the controller board to support the physical control and monitor hardware. Our next task is to provide an interface between the controllers and the equipment which we are to control. In so doing, we will define the hardware I/O assignments for the iSBC 941 processors and for the counters which we will be utilizing. The following paragraphs will deal with the optimization of this configuration.

**** DEVICE ****	**** SIGNAL TYPE ****	**** BOARD ELEMENT ****
WEIGHBELT MOTOR	10 VDC PULSE	ISBC 941
WEIGHBELT WEIGHT	10 VDC PULSE	ISBC 941
WEIGHBELT SPEED	110 VAC PULSE	8259A INTERRUPT
LIQUID VALVE	5 VDC MULTIPHASE	ISBC 941
LIQUID FLOW	10 VDC PULSE	8259A INTERRUPT

Figure 9. Control/Monitor Signals

*** INPUTS ****	*** OUTPUTS ****
GRAVIMETRIC FLOW	ACCUMULATED SOLIDS
LIQUID PERCENTAGE	ACCUMULATED LIQUID

Figure 10. Communication Signals

Controller Interface

Good design practice dictates that we should provide optical isolation between the controller and the external equipment when designing for an industrial environment. The optical isolation is included if we utilize the Intel iCS series of signal conditioning/termination boards. We find that we have two types of digital termination panels available, one for low current, low voltage applications and second for higher current and voltage uses. If we base our choice on the data provided by Figure 8, we will lean toward using the iCS 930 panel for our interface. This board can handle a mixture of signal levels and will support up to sixteen individual lines, providing almost double our needs.

Even a cursory glance at the iSBC 569 controller will provide the knowledge that three edge connectors are utilized to bring the OBS signals from the board. This would indicate that the simplest (and most costly) solution is to use three termination panels. Obviously, we should investigate further before making such a decision. Three possibilities are readily apparent. First, we might

perform some type of re-routing of data lines on the board so as to use only one connector. Second, we can use more than one connector on the ribbon cable and perform a parallel connection of the various lines and choose them so that no duplication of lines results. Finally, we can use some scheme of connecting three cables to the board and use the optional Port C connectors on the termination panel.

The schematic drawings of the IDC indicate that only six of the OBS I/O lines of each processor socket are broken by wire wrap jumper posts. All of the lines so configured are on the Port 2 data lines. Unless we decide to cut etch and add soldered wires, we will not be able to configure our board with this technique. Some further investigation is in order before we can make a decision. The use of a parallel output technique using multiple connectors on a single cable seems to present a feasible approach if we can work out an assignment of I/O which will not cause conflicts. We will begin by building a trial port assignment table in which we will assign the required functions to input/output ports. We will group the inputs and outputs into groups of four to handle the terminator/driver arrangement which is built into the board. This table is shown in Figure 11. We obviously have a small problem. We have

Port	Socket 1	Socket 2	Socket 3	Direction
10		Weight In		In
11				In
12				In
13				In
14				
15				
16				
17				
20	Conv. Mtr.			Out
21				Out
22				Out
23				Out
24			Valve Ph. 1	Out
25			Valve Ph. 2	Out
26			Valve Ph. 3	Out
27			Valve Ph. 4	Out

Figure 11. UPI™ Socket to Terminator Initial Assignments

not yet shown the signals from the conveyor speed and the liquid flow into the on-board interrupt counters. The schematics show that these signals are brought onto the board on the edge connectors but the locations correspond to Port C lines which do not exist on the iCS 930! We have available input lines on the Port 1 connectors but there is no provision to break the signal on the board to route it to the counter interrupts.

If we move on to the third alternative, we find that the interconnection paths caused by tying various lines together cause even greater problems. Either some fact must have been overlooked, or we must consider the use of more than

one terminator board.

Figure 11 indicates that three lines are available on the Port 2 data lines which go to jumper posts and which could be used if they were not part of an output driver of Port 20. If some technique can be found to use these "output" lines as inputs, our problem will be solved. The use of an open collector driver can provide us with the ability to use the line as an input so long as the drivers are turned off! This should be no problem as we can force the outputs to this state either through the appropriate jumpering of inputs or by outputting data to the OBS 1 ports corresponding to these bits. The resulting electrical configuration can be seen in Figure 12.

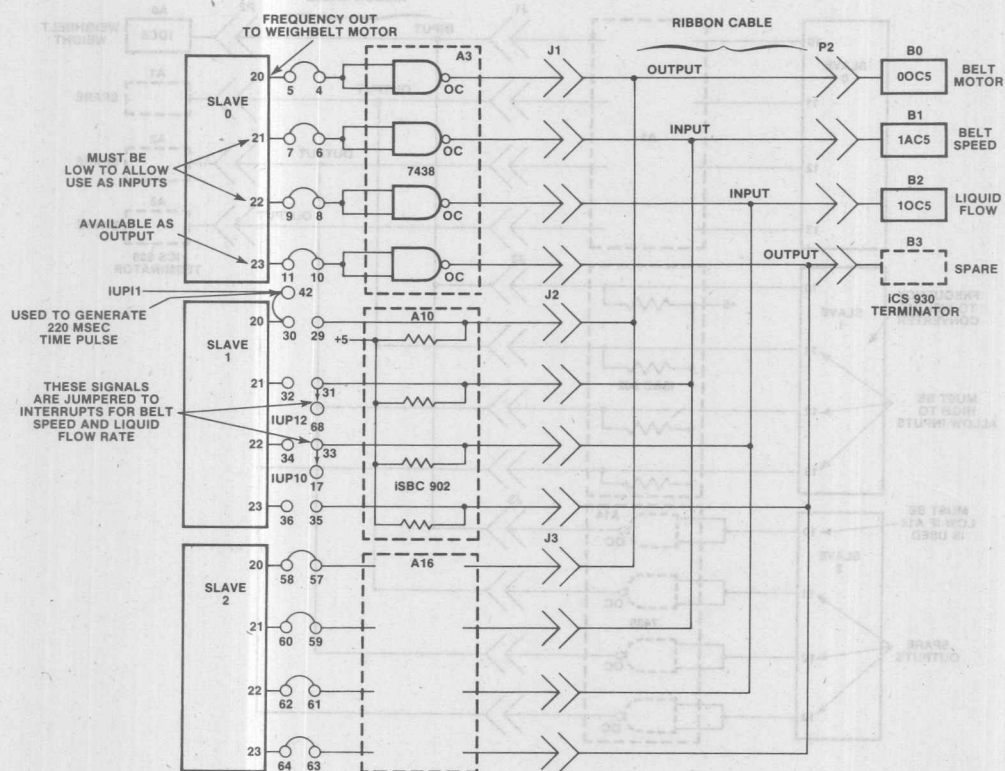


Figure 12. Port Assignments 20-23

Let us examine the implications of performing this interconnection. The physical layout of the board and the use of the terminator/driver sockets causes the I/O lines to be grouped into sets of four data lines. We must choose which of the three iSBC 941 modules will be responsible for supporting each of the lines. In Figure 12, we can see that the belt motor is driven by OBS Socket 1, Bit 20. This requirement has placed output drivers onto data Bits 21, 22 and 23. Our requirement is to provide two signals which can be routed to the counter inputs so we must place a terminator into either socket A10 or A16. We have arbitrarily chosen to use socket A10. The use of the terminators in parallel with the drivers will not create a problem so long as those lines which are used as inputs

have the driver in the high impedance state. This is done by requiring that the output Bits 21 and 22 of the device placed into socket 1 are driven low. Finally, we see that the remaining Bit 23 may be used as a general purpose output line if it becomes required.

The wiring configurations for the remaining connector groupings are shown in Figures 13, 14 and 15. In Figure 13, we see the assignments which can be used for Bits 10, 11, 12 and 13. We have earlier defined that an iSBC 941 processor would be used in a high speed frequency counting mode to determine the weighbelt weight. This device will be placed into socket 2. The use of this mode precludes the use of any general purpose

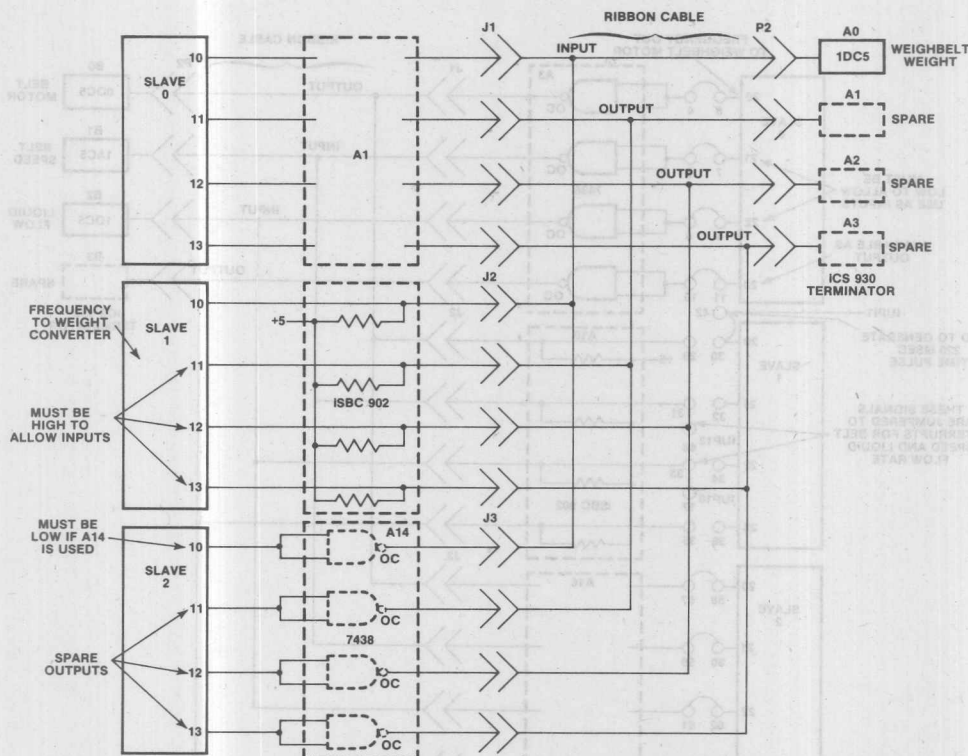


Figure 13. Port Assignments 10-13

input/output operations of the processor if we desire to maintain maximum accuracy of the frequency measurement. We will arbitrarily choose to use Bit 10 as the location of the frequency count input. This will necessitate installing a terminator into the socket corresponding to the processor input. If required, we can install open collector drivers into socket A14 and use the remaining three bits for general purpose outputs. If this is done, care must be taken to assure that Bit 10 of the device which is placed into socket 3 is placed into a low state as was done in the preceding example.

The interconnection scheme for Ports 14 through 17 can be seen in Figure 14. Note that no ports of this group are dedicated to our defined control

functions. These four bits may be used as inputs or outputs as required by the application. For example, we have ignored the fact that actual control loops incorporate solenoids for flow control routing. The unused bits can be used to perform these tasks.

Figure 15 shows the interconnections for the remaining group of bits. There are several features shown on this drawing which should be discussed in some detail. Let us first consider the remaining function which we must implement. This is the control for the liquid valve stepper motor. An iSBC 941 IDP operating in the stepper mode will provide the necessary control functions to drive the motor. Since all four of this group's

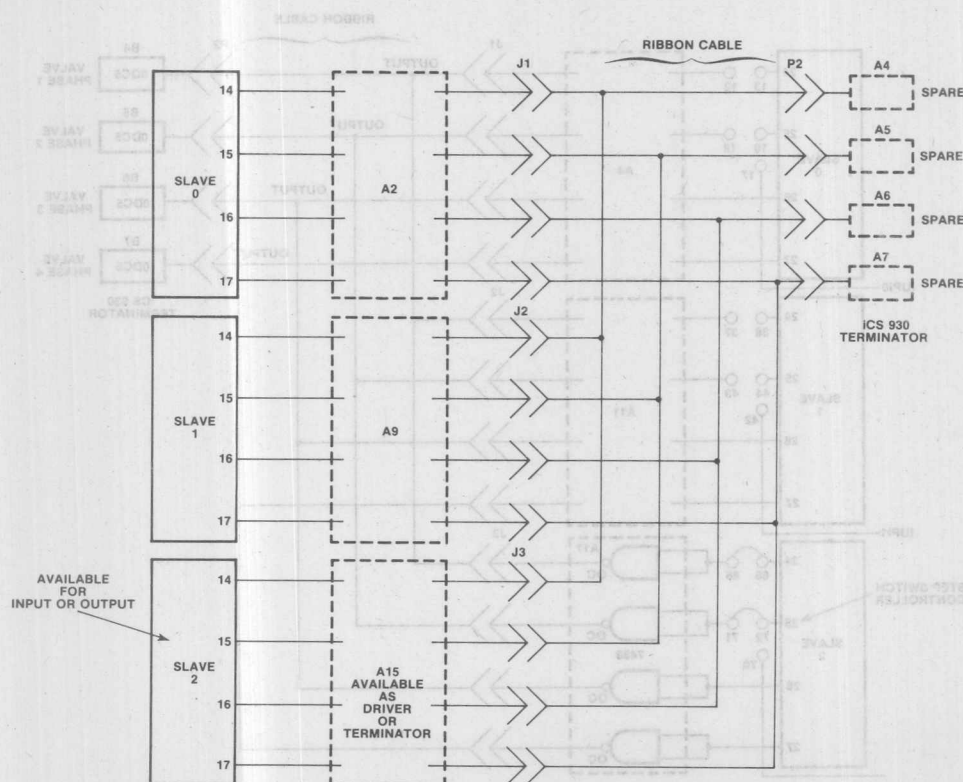


Figure 14. Port Assignments 14-17

data lines are committed to drive the four phases of the stepper motor, there are no other functions available.

An important feature of the iSBC 941 processor is illustrated in Figure 12. This is the ability to enable the processor to generate an interrupt at some point in its operation. We have earlier indicated that we will use the processor in socket 2 (the frequency counter) to provide us with a 200 msec time reference. When the iSBC 941 processor is enabled with an ENFLAG command and is operating in the frequency count mode, it will generate an interrupt on its output line, Port 25. Figure 15 shows how this interrupt can be connected to the host board's internal interrupt input structures.

The hardware configuration has been defined through Figure 14. The actual implementation can be handled through the use of the various wire-wrap jumpers on the IDC. Drivers and terminators can be installed as indicated in the preceding discussion.

VI. SOFTWARE CONFIGURATION

As with most computer controlled systems, the actual implementation of the task is handled with software. In older designs and in many mini-computer systems, this task has become formidable and has resulted in cost over-runs and schedule delays. Intel provides many tools for use by the designer to prevent this type of problem and to assist him in easily creating a workable and well

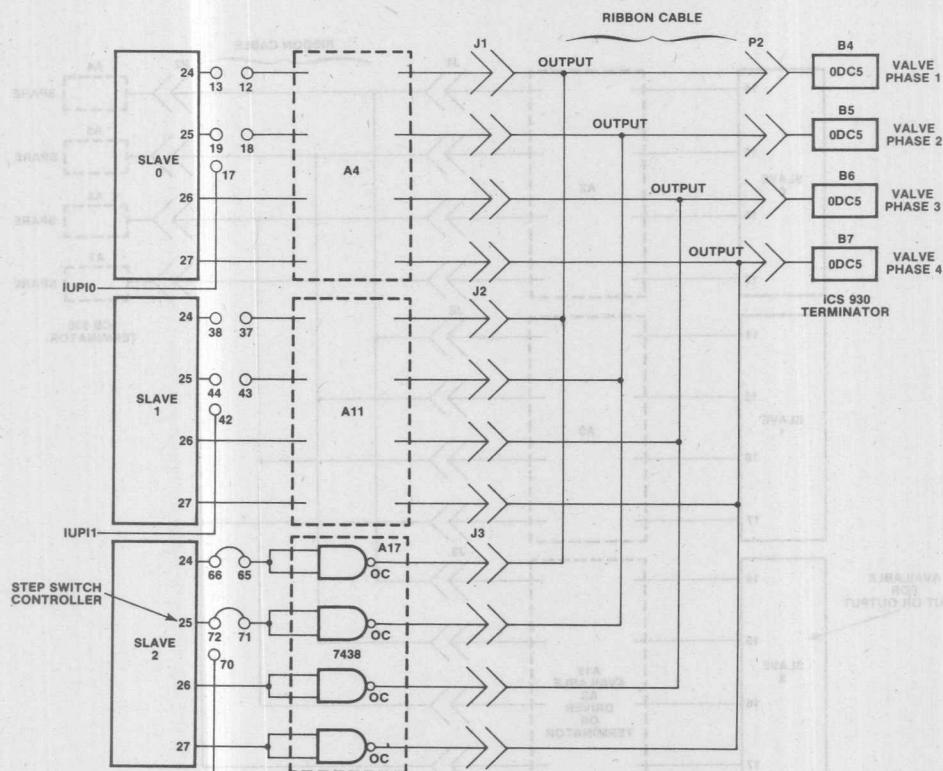


Figure 15. Port Assignments 24-27

documented software configuration. Let us look at some of these tools in more detail and consider how their use will help us to write our programs easily and quickly.

High Level Programming Languages

A valuable tool, which Intel provides the designer of small control systems, is the ability to program even the smallest systems using a high level programming language, PL/M-80. This language offers relatively efficient and structured, programming capabilities. It has been determined that PL/M-80 users can expect to use between 1.1 to slightly more than 2 times as much program memory as would be used for the same task written in assembly language. At the same time, the programmer's time to code a task will be considerably less than if he were to use assembly language. The PL/M-80 Programming Manual indicates that the language is highly structured and lends itself very well to handle logical type operations. Its weakness in handling complex mathematical computations is compensated by the ability to combine the user application software with packaged Intel support software.

Fundamental Support Packages

The Intel 8080/8085 Fundamental Support Package (FSP) provides a package of application subroutines and functions which can be called from programs written in either assembly language, PL/M-80, or in FORTRAN-80. It uses a standard set of data structures and a unified status and error reporting scheme. Nine major groups of operations are fully supported by this package. These are:

1. A primitive fast string handling and integer arithmetic capability without error reporting.
2. A binary integer arithmetic package which performs operations on both signed and unsigned integers of various lengths in binary representation.
3. The floating-point arithmetic package which provides operations on floating point numbers in four formats: single precision, single-precision extended, double precision, and double-precision extended.

4. The decimal arithmetic routines which perform integer and fixed point computations on numbers which are stored as strings of ASCII characters.
5. A string handling section which contains routines to transform strings and to extract and insert substrings. A routine for scanning of general input and one for formatting of general output are included.
6. Routines for number conversion, for numeric I/O transformation of data from one format to another, input scanning of numeric strings, and formatting of numeric strings for output are also available.
7. The floating point transcendental function section provides trigonometric, exponential, and other transcendental functions.
8. The statistics routines compute the mean, variance, and standard deviation of one group of statistical data, and the covariance and correlation factor of two groups of data.
9. Finally, the PID procedures provide the user with a version of the classical Proportional, Integral, Derivative control algorithm.

Clearly, the use of the FSP support programs enhance the logical PL/M-80 program operations.

Host/Slave Relationship

Before we proceed with our development, we should take some time to examine the relationship between our iSBC 569 IDC and other controllers which may be installed in the system. The utilization of intelligent slave boards provides the capability to develop control concepts to an extremely high level if certain guidelines are followed. We will therefore assume that the control solution which we are developing will be but a part of an over all control concept which utilizes multiple controllers sharing common resources.

This concept allows us to develop control algorithms for each sub-process within our overall control system. This development can provide independent design and implementation of each process. A host processor can be used to provide any required inter-process communication tasks and to provide the operator interface. We have previously indicated that the operator interface will provide some means to adjust the weight

feeder setpoints and the liquid ratio. It should also allow the operator to display the current status of the process. Since these operator interface functions are but a part of the overall control functions, the interface should be programmed such that maximum flexibility can be gained through its use. Fortunately, such an interface is available using Intel's RMX/80 BASIC-80.

RMX/80 BASIC-80 Interpreter

The RMX/80 BASIC-80 Interpreter is a high level language interpreter with extended disk capabilities. It operates on iSBC 80 Single Board Computers and allows the interpretation of BASIC-80 source code into an internally executable form. Many other features are available and many configurations are possible depending upon the exact system requirements (refer to the *BASIC-80 Reference Manual*, 9800758).

Maximum utilization of the operator interface with a minimum of development time can be achieved with the preconfigured version of the software/hardware package. This will provide us with complete disk I/O capabilities and the ability to easily program and maintain any programs which may become necessary to implement the interface. The actual implementation of the interface will be done later, after we have defined the control task.

Software Tasks

The task of preparing the software can be broken down into three major groupings or tasks. These are defined to be:

Prepare the Software Drivers.

This involves defining the relationships between the control algorithm parameters and the input/output hardware devices and creating software to implement these definitions.

Prepare the Control Algorithm.

This will involve developing a control algorithm which defines the relationships between the various system parameters. This

algorithm will draw heavily upon the resources of the FSP programs and the software drivers which relate the parameters to the physical hardware.

Finally, the operator interface must be defined which will relate the parameters used in the control scheme to other controllers and to the operator. This will allow the control task to interact in such a manner as to provide a meaningful element of the overall control concept.

VII. SOFTWARE DRIVERS

Before developing the actual control algorithm, we must create the drivers which communicate with the three iSBC 941 processors in their assigned operating modes. We will define two driver sections for each processor, one to handle the initialization, and a second to provide the ongoing communications as required by the control algorithm program.

Motor Speed Control Processor

The first processor which we will discuss is to be located in slave socket number 0 and will be used to produce a variable frequency output. Let us consider in some detail how this can be accomplished using an iSBC 941 Processor. First, consider the task of initializing the device to the primary function operating mode, FREQ.

Referring to the *iSBC 941 Industrial Digital Processor User's Guide*, we find that the initialization requires the sequence of commands and data shown in Figure 16. We will identify the meaning of each of these terms and create a software

Description	Command/Data
Request INIT	C
FREQ Select	D
Scale Factor	D
Output Enable	D
Initial State	D
P20 Delay	D
P20 Period	D
Request PAUSE	C

Figure 16. FREQ Initialization

program which will handle the required initialization of the processor. The purpose and use of the various commands to the processor are well defined in the user's guide and will not be repeated here.

The first byte of data, which must be sent following the initialization command, is the data byte signifying that the operational mode is to be the frequency output. This is defined in the manual as being equal to the data byte "0B5H" or "035H" as expressed in the hexadecimal numbering system. The choice of values to be sent is dependent upon our desire to utilize the internal or external time reference period for the operations. If we utilize the internal time reference, our minimum increment or resolution of operations will be 86.72 microseconds.

To determine if this speed is adequate for our frequency generator, we must consider the impact that this resolution has on the output. A 550 hertz signal has a period of 1.82 milliseconds. If we increase this period by the 86.72 microsecond time reference, we find that the next increment in the frequency generators output will be approximately 372 hertz. This resolution is certainly not adequate to meet the motor control requirements! We should consider using the external clock to provide the time reference. One of the 8253 Interval Timers on the iSBC 569 board can be used to generate a reference time. If we arbitrarily choose to use a 10 microsecond reference to the IDP, we find that the worst case resolution for the 550 hertz signal becomes about 4 hertz. This is certainly within our requirements of motor control. The primary function signal should then be sent as a "0B5H".

The second byte is used to establish a scale factor for the processor. This scale factor is used to generate the basic time increment which can be used to establish the frequency output; that is, the minimum time increment which can be used to establish a period or pulse width will be the scale factor times the reference time period.

In our case, because of the wide frequency output range, we cannot specify the scale factor at initialization (later data will show the need for

multiple scale factor ranges). We will then only need to send some arbitrary value at initialization to allow the processor to complete its initialization sequence.

The Output Enable data byte is used to select which of the Port 2 output bits are to be used to generate the output signals. The hardware configuration established earlier placed the output onto Bit 0 of the port, so this data byte shall be specified as a byte having only Bit 0 set to a logical one or equal to 01H.

The Initial Output parameter specifies whether each bit selected as an output by the output enable byte is to be initially set to a logical one or zero when the processor is first enabled. For this application, it really does not matter, but we will arbitrarily pick the state to be equal to zero. The byte will be defined as being set to 00H.

The Delay parameter is used to define the waveform which will be generated and specifies the number of time increments which must elapse before the waveform will change states. Rather than to constantly vary the delay to maintain a square wave output, we can choose an arbitrary value of one time increment before changing state. The output will have a varying duty cycle as the frequency changes. This should cause no problems for the translator driving the weighbelt motor. The byte will be defined as being set to a value of 01H.

Finally, the Period of the waveform must be chosen. Again, this parameter will be changed according to the desired frequency, so only an arbitrary value need be sent. Indeed, since this is the last parameter, the value could be omitted entirely by sending the PAUSE command in its place.

The initial data definition can be defined using PL/M-80 language conventions as a block of six bytes as shown in Figure 17.

The actual communications between the host processor on the iSBC 569 board and the IDP utilizes the protocol explained in previous sections of this note. The status register of the IDP will be tested for the bit signifying that the input buffer

```

/* DECLARATION OF ISBC 941 #0 INITIALIZATION DATA */
22 1 DECLARE FREQ LITERALLY '0B5H';
23 1 DECLARE SF LITERALLY '000H';
24 1 DECLARE OUTPUT$ENABLE LITERALLY '001H';
25 1 DECLARE INITIAL$STATE LITERALLY '000H';
26 1 DECLARE DELAY LITERALLY '001H';
27 1 DECLARE PERIOD LITERALLY '000H';

/* DECLARATION OF ISBC 941 PRIMARY DATA */
34 1 DECLARE INIT$TABLE(6) BYTE DATA (
    FREQ,
    SF,
    OUTPUT$ENABLE,
    INITIAL$STATE,
    DELAY,
    PERIOD );

```

Figure 17. Initial FREQ Data Field

full is not set. This will indicate that the device is ready to accept either a command or a data byte. The command to request a primary function will be sent. At this point, the processor will be expecting a series of data bytes as specified by the

function being selected. A "Do Loop" can be used to effectively transmit this data to the device. The program to perform this function is illustrated in Figure 18.

```

/* REQUEST PRIMARY FUNCTION */
44 2 DO WHILE ( (INPUT (UPI$0$STATUS) AND IBF) < > 0);
45 3 END;
46 2 OUTPUT (UPI$0$COMMAND) = INITPF;

/* LOAD INITIAL PARAMETERS */
47 2 DO I=0 TO 5;
48 3 DO WHILE ( (INPUT (UPI$0$STATUS) AND IBF) < > 0);
49 4 END;
50 3 OUTPUT (UPI$0$DATA)=INIT$0$TABLE(I);
51 3 END;

/* TERMINATE PARAMETER LOADING */
52 2 DO WHILE ( (INPUT (UPI$0$STATUS) AND IBF) < > 0);
53 3 END;
54 2 OUTPUT (UPI$0$COMMAND)=PAUSE;

/* START FREQUENCY FUNCTION */
55 2 DO WHILE ( (INPUT (UPI$0$STATUS) AND IBF) < > 0);
56 3 END;
57 2 OUTPUT (UPI$0$COMMAND)=LOOP;

```

Figure 18. IDP Initialization

When all required data parameters have been sent, the data portion of the initialization is terminated by sending a PAUSE command as shown in Figure 18. Note how, in each case before data or a command is sent, we wait until the input buffer is empty. Finally, the initialization is completed when we have sent the LOOP command. The processor will now be generating an output frequency as specified by the parameters.

Remember that, according to our earlier discussion and as we have shown in Figure 12, the unused output ports should be set to a logical low condition to allow the use of those lines as inputs to carry additional data into the controller. This should be done as a part of the initialization process. The secondary utility command, CLRP2 is used for this purpose. This process is illustrated in Figure 19.

We should next direct our attention to establishing a software interface which will take the desired

weighbelt speed term and convert it to a frequency output suitable to drive the motor translator. We know that this will involve selecting a particular scale factor and period term which will generate the correct waveform. Previously, we established that, for a maximum frequency of 550 hertz, we need to establish a period of 1.82 milliseconds. Many combinations of Scale Factor and Period parameter will generate this time interval. Ideally, the smallest increment of change can be established by setting a constant period and modifying the scale factor. If we make some calculations, we will find that the fact that the scale factor is a byte value (giving us a range of between 0 and 255) limits the frequency range which can be produced using any one value for a period. It seems that we will be forced to vary both the period and the scale factor as a function of the desired frequency.

In Figure 20, we have plotted the frequency output for various values of Scale Factor and Period. Our

```

/* SET UNUSED BITS TO ALLOW EXPANSION */
59 2 DO WHILE ( (INPUT UPI$0$STATUS) AND IBF) < > 0);
59 3 END;
60 2 OUTPUT (UPI$0$COMMAND)=CLRP2;

61 2 DO WHILE ( (INPUT (UPI$0$STATUS) AND IBF) < > 0);
62 3 END
63 2 OUTPUT (UPI$0$DATA)=INITIAL$OUTPUT;

```

Figure 19. Secondary Utility Command

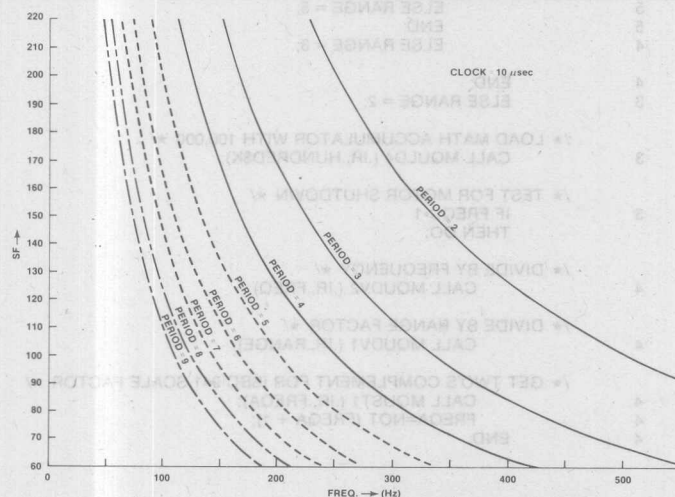


Figure 20. Frequency Vs. Parameters

intent is to maintain the highest resolution possible for the desired output range of 50 to 550 hertz. Choosing four period base parameters will provide us with acceptable waveform generation characteristics. We will choose the data sets of Figure 21 based upon the data shown in Figure 20.

The Period can be determined by examining the desired frequency range. The scale factor can be calculated from the equation:

$$SF = 10,000 / ((\text{FREQUENCY}) \times (\text{PERIOD}))$$

Again, the PL/M-80 language program to implement the interface between the host and the IDP is easily constructed. For example, Figure 22 provides the code which will be required to determine the appropriate Period parameter and also illustrates the use of FSP programs to handle

the mathematical calculations required to determine the corresponding scale factor.

The principles above can be expanded into a complete interface package to offload the host processor of the need to generate the frequency waveform to the translator of the weighbelt motor. The complete program for the processor can be found in Appendix A.

Weight Input Processor

The second use of an iSBC 941 Processor is to provide the capability of converting the high frequency inputs from the weight sensor of the weighbelt into a digital value equivalent to the actual weight on the belt. This frequency to digital conversion can be easily accomplished by the use of the Primary Function, FCOUNT.

Frequency	Period	Scale Factor	Resolution
50 to 165 Hz.	9	221 to 67	3 Hz.
166 to 225 Hz.	5	121 to 89	3 Hz.
226 to 285 Hz.	3	147 to 117	3 Hz.
286 to 550 Hz.	2	175 to 91	6 Hz.

Figure 21. FREQ Output Ranges

```

57 3  /* COMPUTATION OF FREQUENCY RANGE */
      IF FREQ < 285
59 4  THEN DO;
      IF FREQ < 226
61 5  THEN DO;
      IF FREQ < 166
63 5  THEN RANGE = 9;
64 5  ELSE RANGE = 5;
65 4  END;
      ELSE RANGE = 3;

66 4  END;
67 3  ELSE RANGE = 2;

68 3  /* LOAD MATH ACCUMULATOR WITH 100,000 */
      CALL MQULD4 (.IR.,HUNDRED$K);

69 3  /* TEST FOR MOTOR SHUTDOWN */
      IF FREQ > 1
      THEN DO;

71 4  /* DIVIDE BY FREQUENCY */
      CALL MQUDV2 (.IR.,FREQ);

72 4  /* DIVIDE BY RANGE FACTOR */
      CALL MQUDV1 (.IR.,RANGE);

73 4  /* GET TWO'S COMPLEMENT FOR iSBC 941 SCALE FACTOR */
74 4  CALL MQUST1 (.IR.,FREQA);
75 4  FREQA=NOT (FREQA + 1);
      END;

```

Figure 22. Period and Scale Factor Computations

The FCOUNT Primary Function is selected by sending the INITPF command followed by four parameters. The process is identical to that which was used in the previous example when we established the FREQ function. In this case, the sequence is described in the manual as is shown in Figure 23.

Description	Command/Data
Request INIT	C
Select FCOUNT	D
Input Select	D
Output Enable	D
Sampling Interval	D
Request PAUSE	C

Figure 23. FCOUNT Initialization

Let us examine the derivation of the terms which must make up the data table which will be transmitted to the processor in order to initialize it. The FCOUNT function does not allow the use of an external clock so we have no option as to which command will be sent to select this function. It is defined to be equal to 33H. This becomes the first element of the byte array used to contain the initial data.

The Input Select parameter describes which of the Port 1 inputs are to be measured. If we refer to Figure 13, we can see that a hardware assignment of Port 10 has been made for this function. This assignment corresponds to bit 0 of the parameter being set to a value of 1. The byte value for this parameter then becomes 01H.

The Output Enable byte is used to enable an output port corresponding with the input to change states when the Sampling Interval time has elapsed. Our system has a requirement to operate the control algorithm once each 200 milliseconds and we have previously indicated that the frequency counter would be used to establish this time interval. If the output is enabled and connected to an interrupt line, it will provide our system with the required pacer clock. The output bit from Port 20 will then be enabled to provide the interrupt. The parameter for this byte will be set to the same value as the Input Select and becomes 01H.

The Sampling Interval will establish the time interval to be used when sampling the input frequency. This time interval should be set to 200

milliseconds for our application. The parameter is then calculated from the equation:

$$\text{INTERVAL} = (\text{SAMPLE PERIOD}) / (0.02222) \\ \text{OR} \\ \text{INTERVAL} = (0.200) / (0.02222) = 9$$

The correct sampling interval for our control system should be set to a value of 09H.

A similar procedure can be used to send this data to the processor. The actual code used to implement the system can be found in Appendix A. Note that the unused bits of the device have been set to a predetermined value as was indicated by our hardware design of Figure 13.

Once the processor has been initiated and is performing its function, we need only wait until the device signals us that the 200 millisecond time interval has passed and that it is ready with the belt weight. When this interrupt occurs, we will read the data and perform our control functions. An interface must be established between the control algorithm and the processor which enables it to receive a value which represents the actual weight.

The total count received by the processor is available as a sixteen bit count made up of two eight bit bytes. The use of the Secondary Utility Commands, Read FCOUNT Measurements (RDFC0-RDFCF) allow the two bytes to be transferred into the host processor. We are using the first counter so we will use the corresponding commands, RDFC0 and RDFC1. An example of the procedure to read one of the count bytes can be seen in Figure 24.

The counter can be commanded to begin its next sample period by issuing a LOOP command to the processor. The two data bytes can be combined to form a 16-bit word and the resultant value divided by 2 to form a weight value. The division by two to obtain weight is required since the count range from 0 to 2000 corresponds to a weight of between 0 and 10.00 pounds; thus, each count has a value of 0.005 pounds. The integer numbers used in the control algorithm are fixed point with an implied scale factor of 100. The division by two provides a result which meets the criteria.

```

106 2 /* GET INPUT COUNT LOW BYTE */
107 3 DO WHILE ( (INPUT (UPI$1$STATUS) AND IBF) < > 0);
108 2 END;
      OUTPUT (UPI$1$COMMAND) = RDFC0;
109 2 DO WHILE ( (INPUT$1$STATUS) AND OBF) = 0);
110 3 END;
111 2 LCOUNT = INPUT (UPI$1$DATA);

```

Figure 24. FCOUNT Read Procedure

Appendix A provides the complete listing of the code which was used to interface with the processor assigned to the primary function, FCOUNT.

Stepper Motor Control Processor

The third example of utilizing the iSBC 941 Processor in an industrial application is provided by the processor installed into OBS socket 2. This device is used to drive a stepper motor which, in turn, controls the liquid valve position. Again, we will break the discussion into an initialization and an interface operational mode.

We find that the User's Guide indicates that initialization to the STEPPER Primary Function is performed by sending the INIT command followed by up to 21 data bytes. Figure 25 provides the table which shows the necessary parameters for this mode.

The technique used to place the processor into the desired function is the same as we have seen with the two other processors so we will not spend time dealing with the communications sequence. Instead, we will examine the techniques which can be used to determine the values of the initialization parameter bytes.

STEPPER is requested by sending a data byte of either 17H or 97H following the INIT command. Remember that the significance of setting bit 7 of the data high is to request that an external clock be used by the processor. There is no reason to use an external clock for our application, so we can choose a function request byte of 17H.

The remainder of the data is used to define the waveforms which are necessary to drive the stepper motor. We will derive the values for these parameters by beginning with the manufacturer's data sheet and moving until we have determined the correct value for each byte of data.

The motor chosen for this application utilizes four phases to drive the shaft. The data sheet provided

Description	Command/Data
Request INIT	C
Select STEPPER	D
Select Scale Factor	D
Output Enable	D
Output Polarity	D
Common Period	D
P20TRAN1	D
P20TRAN2	D
P21TRAN1	D
P21TRAN2	D
P22TRAN1	D
P22TRAN2	D
P23TRAN1	D
P23TRAN2	D
P24TRAN1	D
P24TRAN2	D
P25TRAN1	D
P25TRAN2	D
P26TRAN1	D
P26TRAN2	D
P27TRAN1	D
P27TRAN2	D
Request PAUSE	C

Figure 25. STEPPER Function Initialization

information for both a Four-Step Input Sequence (1.8 degrees per step) and for an Eight-Step Input Sequence (0.9 degrees per step). We will use the 1.8 degree step angles for our example and application. The data provided by the manufacturer is shown in Figure 26. The first task is to convert the switch state diagram into a desired waveform for each of the four phases. This has been done in Figure 27.

Beginning with Scale Factor, let us determine the required data parameters which will yield a stepper controller compatible with our motor. The Scale Factor will provide the minimum time period for one step to take place. The minimum time which we can specify is a function of both the motor characteristics and of the TRP for the primary function, STEPPER. The minimum TRP is determined by referencing the IDP User's Guide for the desired function. In this case, it is found to be $325 + (13 \times B)$ where B is the number of phases

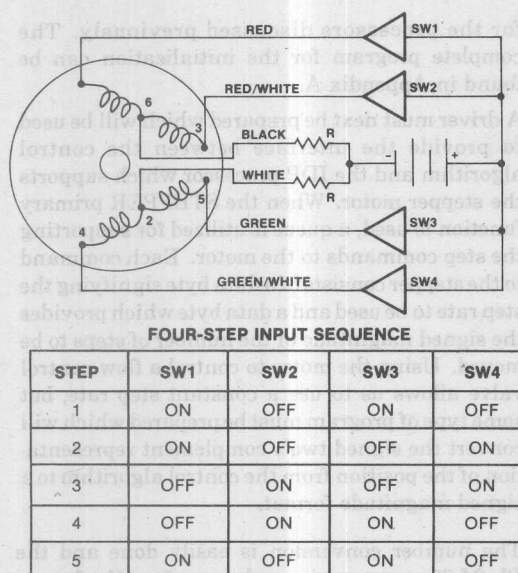


Figure 26. STEPPER Motor Input Sequence

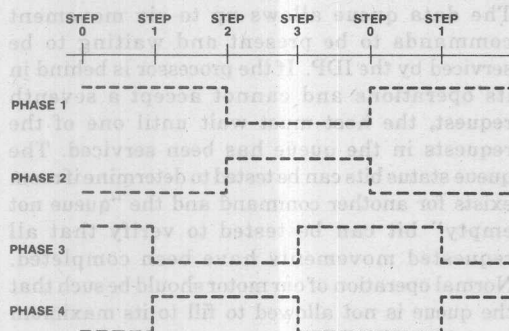


Figure 27. STEPPER Motor Waveforms

which are used. The result will be expressed in terms of processor cycles and can be converted into time by multiplying by 2.71 microseconds per cycle. This works out to be:

$$325 + (13 \times 4) = 377 \text{ PROCESSOR CYCLES}$$

OR

$$377 \times 2.71 = 1.021 \text{ MILLISECONDS}$$

Now, let's examine the minimum time which can be utilized by the stepper motor. This is given in the manufacturer's data sheets as being 2.86 milliseconds for the motor which we have chosen to

use. This value must be used to compute the Scale Factor for this application. The Scale Factor is computed by dividing the minimum step time by 86.72 microseconds or:

$$SF = 2.86 \text{ MILLISECONDS} / 86.72 \text{ MICROSECONDS} = 33$$

This number is entered into the processor using two's complement which becomes equal to 0DFH.

The Output Enable is used to specify which of the eight possible control outputs are to be used to control the motor phases. The motor phase assignments to I/O ports was made in Figure 15 and indicates that Ports 24 through 27 will be enabled for the primary function. Setting the corresponding bits provides a parameter to be sent to the processor of 0F0H.

The rest of the parameters deal with providing a definition of the waveforms generated in Figure 26 to the processor. The following paragraphs deal with the operations required to convert the graphic representation into data parameters.

Each phase must be initialized to an initial output state which corresponds to the signal level shown for Step 0 of Figure 27. A "1" will be placed into the bit corresponding to each of the port's output bits which are to be in a logical one state upon

reaching step 0. We see that Bits 24 and 26 are set corresponding to phase 1 and 3. The data byte for Initial Output is thus defined to be 050H.

The Period parameter for a stepper motor function corresponds to the number of steps which are defined in the motor's step sequence. Our example uses a four step sequence so the Common Period will be set to a value of 04H.

The remainder of the initialization parameters define the transitions of each of the phases. This involves the examination of the waveform and noting the points at which the output level changes. This data can be input to allow the device to accurately produce the control waveforms for any stepper motor control mode. We are not using the first four output bits so the transition definitions for these outputs is meaningless and will be output as zeroes. The waveform for output Port 24 shows a transition at steps 1 and 3. The parameter for the first transition of Port 24, P24TRAN1 is defined to be 00H. Likewise, the second transition, P24TRAN2 is set to a value of 02H.

The technique used above can be continued to define the constants, P25TRAN1 and P25TRAN2 as being the same as for Port 24 or 00H and 02H respectively.

The transitions for the phases driven from Port 26 and 27 can be seen to occur at steps 1 and 3 so the data for those parameters can easily be seen to be set to 01H and 03H for each port.

The initialization table can be sent to the processor using the same techniques as were used

for the processors discussed previously. The complete program for the initialization can be found in Appendix A.

A driver must next be prepared which will be used to provide the interface between the control algorithm and the IDP processor which supports the stepper motor. When the STEPPER primary function is used, a queue is utilized for supporting the step commands to the motor. Each command to the stepper consists of a data byte signifying the step rate to be used and a data byte which provides the signed magnitude of the number of steps to be moved. Using the motor to control a flow control valve allows us to use a constant step rate, but some type of program must be prepared which will convert the signed two's complement representation of the position from the control algorithm to a signed magnitude format.

The number conversion is easily done and the PL/M-80 programming code to perform the format change is shown in Figure 28.

The data queue allows up to six movement commands to be present and waiting to be serviced by the IDP. If the processor is behind in its operations and cannot accept a seventh request, the host must wait until one of the requests in the queue has been serviced. The queue status bits can be tested to determine if room exists for another command and the "queue not empty" bit can be tested to verify that all requested movements have been completed. Normal operation of our motor should be such that the queue is not allowed to fill to its maximum capacity.

```

/* SUPPORT CONVERSION TO SIGNED MAGNITUDE NUMBER */
141 3 IF POSITION > 127
    THEN DO;
143 4 /* GET MAGNITUDE OF MOVEMENT */
    POSITION = 256 - POSITION;
144 4 /* SET SIGN FOR CCW ROTATION */
    POSITION = POSITION OR REVERSE;
145 4 END;

```

Figure 28. Number Format Conversion

The code which is required to test the queue and to send a stepper movement request is shown in Figure 29. The complete code can be seen in Appendix A.

VIII. APPLICATION SOFTWARE

Having developed the software which is required to support the Industrial Digital Processors, we can now devote our time to the task of implementing the application software and of handling any programs which are required to support functions unique to the host iSBC 569 board. This software can be grouped into two general categories, initialization programs, and control algorithm programs.

Initialization Programs

The initialization of the iSBC 569 involves setting up the required configuration of interrupt handling and of the devices which are installed into the slave sockets. For the purposes of this application, we will include some system diagnostic capabilities within the process. These routines will be executed each time a RESET or a POWER-UP occurs. Only the highlights of the code used will be presented in detail; however, the complete listings of the initialization programs can be found in Appendix A by referring to the BCKGND Program listing.

A unique feature of using the iSBC 941 processors is their ability to provide, upon request, an

identification code. The initiation diagnostic program takes advantage of this fact by interrogating each processor and verifying that the correct ID code is returned. If any of the processors have failed catastrophically or if the internal data bus of the host board has failed, the program will provide an indication of this fact.

Each of the slave processors has, associated with it, an individual hardware reset line which is under the control of the host. A reset or power up condition will cause the control lines to reset to the state which hold each slave in a reset state. Before any slave can be used, it's associated reset line must be de-activated. This is done by sending a logical one to the corresponding bit of the Reset Latch. Other bits of the Reset Latch can be used to illuminate the on-board LED or to generate an interrupt to another board on the Multibus data bus.

A special PL/M-80 command is utilized to disable the reset interrupts of the 8085A host processor. Execution of this command will allow all servicable interrupts to enter via the 8259A Interrupt Controller. The command which will mask off the unused interrupt structure is shown in Figure 30.

The initialization process must also initialize the FSP Integer Record. This will allow the use of the math support routines which will be required to support the control algorithm.

```

146 3 /* VERIFY THAT QUEUE SPACE IS AVAILABLE */
147 4 DO WHILE ( (INPUT (UPI$2$STATUS) AND QF) < > 0);
      END;

148 3 /* REQUEST DESIRED STEP RATE */
149 4 DO WHILE ( (INPUT (UPI$2$STATUS) AND IBF) < > 0);
150 3 END;
      OUTPUT (UPI$2$DATA) = STEP$RATE;

151 3 /* REQUEST STEPPER MOVEMENT */
152 4 DO WHILE ( (INPUT (UPI$2$STATUS) AND IBF) < > 0);
153 3 END;
      OUTPUT (UPI$DATA) = POSITION;

```

Figure 29. STEPPER Movement Request

```

34 1 /* MASK OUT THE RESET INTERRUPTS OF THE PROCESSOR */
      CALL S$MASK (MASKS);

```

Figure 30. PL/M-80 Sim Instruction

Control Algorithm Programs

The program which actually handles the control algorithm for the two loops can be found in Appendix A, MAIN\$CONTROL. The flow of the program is straightforward and can easily be followed by reading the listing. The operations are primarily handled by the use of repeated calls to the FSP integer math routines and to the processor interface modules which we have previously generated.

It is beyond the scope of this application note to dwell upon the techniques which were used to generate the PID control routine; this aspect will be covered in a future application note.

Limits were placed upon the control outputs so that the signals to the processors would not exceed the physical limits of the external devices. For example, the frequency range is limited to range between 0 and 550 to correspond with the operating range of the weighbelt as we have defined it. The limits upon the liquid control valve have been set at plus and minus 10 steps since this is the maximum distance which the stepper motor can travel in any one 200 millisecond time period; increasing the possible count could result in filling the queue. This could cause the 200 millisecond time to be extended if we had to wait for the queue to empty.

Master Processor

A complete control solution to the weighbelt feeder and the liquid applicator has now been developed. The process is stand alone and resides entirely upon a single board. It can operate without requiring any access from the MULTIBUS bus, thus freeing the bus for other control, monitoring or supervisory duties.

The system developed for this application note requires a setpoint for the mass flow and a liquid ratio be provided to the control system. This information would normally be supplied by some type of bus master device. We have chosen to use the pre-configured RMX/80 BASIC-80 Interpreter to perform this task. A simple program needs to be prepared which will allow adjustment of the setpoints and monitoring of the operation of the control system.

Using BASIC will provide full disk I/O capabilities to the operator. Communicating with the

system through a CRT terminal, he can write and execute programs which use the resources of the system disk or of any of the controllers which may be present on the bus.

Two programs are required to perform this task. Since they are written in BASIC, they may easily be modified or expanded if the need should ever arise. Indeed, other programs could be written to perform other tasks, such as optimizing the control parameters.

In both programs, the parameters involved with the control operation are accessed by using the PEEK and POKE instructions. Remember that the iSBC 569 controller allows the on-board memory to be made available to other devices on the bus through the dual port mechanism. In our application, this has been done by jumpering the board such that the on-board memory beginning at location 8000H can be accessed on the bus at location 2000H. This mapping was done since the memory locations at 2000H are not used by BASIC unless requested to do so. A byte of data which is at location 827EH on the controller can be read by performing a PEEK of location 227EH. Some of the memory assignments for the controller have been shown in Figure 31.

MOD MAINCONTROLMODULE		
829 FH	SYM	MEMORY
823 3H	SYM	PRLQ
825 FH	SYM	CONSTANTS1
00D CH	SYM	BOUNDS2
00E 6H	SYM	TIMEINTERVAL
827 AH	SYM	LIQUIDFLOW
00E 8H	SYM	DISTREV
828 0H	SYM	MASSFLOW
828 5H	SYM	LIQUIDVALVE
828 8H	SYM	DUMMY
00E FH	SYM	ZERO
01ADH	SYM	PIDRUN
81F 7H	SYM	IR
825DH	SYM	LIQCOUNT
826 8H	SYM	CONSTANTS2
00E 4H	SYM	CONTROL1
827 7H	SYM	BELTSPEED
827 CH	SYM	MASSSETPOINT
00E 9H	SYM	CONVLNGTH
828 2H	SYM	BELTCONTROL
828 7H	SYM	SYSTEMRUNNING
828 AH	SYM	ICW
3F0 0H	SYM	JUMPTABLE
820 9H	SYM	PRCV
825 EH	SYM	BELTCOUNT
00D 4H	SYM	BOUNDS1
00E 5H	SYM	CONTROL2
827 8H	SYM	BELTWEIGHT
827 EH	SYM	SETPOINT
00E AH	SYM	SIX
828 4H	SYM	LIQUIDRATIO
00E BH	SYM	ERRORFIELD
00E DH	SYM	THOUSAND
00F 1H	SYM	INITIATION

Figure 31. Selected Memory Location Assignments

The first program involves setting up the two control parameters and handling the control flag which causes the process to start and to stop. This program can be found in Figure 32.

```

10 REM THIS PROGRAM IS USED TO INPUT SETPOINTS
15 REM TO THE LIQUID CONTROL SYSTEM.
20 POKE 02287H,0
25 INPUT "ENTER MASS SETPOINT-";MS
26 IF MS > 1200 THEN 25
30 MS=CINT(MS*10/60)
35 H=INT(MS/256)
40 L=CINT(MS-H*256)
45 POKE 0227EH,L
50 POKE 0227FH,H
55 INPUT "PERCENT LIQUID-";LR
60 LR=CINT(LR)
65 IF LR > 127 THEN 55
70 POKE 02284H,LR
75 POKE 02287H,1
80 RUN "STATUS"

```

Figure 32. Basic Program for Parameter Initialization

Upon completion of the initialization program, a second program provides a display of the system operation. This program could have been an optional program which is only called when the operator desires to view the system operation. A program which provides a snapshot of the system operation is shown in Figure 33. Again, the program is interactive with the operator and can easily be modified at any time to reformat or display additional information.

IX. CONCLUSION

The purpose of this application note has been to demonstrate some of the techniques which can be used to provide a control system design solution using an intelligent slave concept. This has been done and the system has been constructed and has been found to operate as the design specified. The Intelligent Slave Concept does provide a single board solution to distributed control and certainly off-loads the master processor of control duties.

PROGRAM NAME: STATUS

```

10 I=PEEK(0227EH)
20 H=PEEK(0227FH)
30 MS=((256*H)+L)*60/10
40 L=PEEK(02278H)
50 H=PEEK(02279H)
60 WT=((256*H)+L)/100
70 L=PEEK(022890H)
80 H=PEEK(02281H)
90 AM=((256*H)+L)*60/10
100 MT=PEEK(02294H)
110 LR=((PEEK(02284H))/100)
120 LS=AM*LR
130 L=PEEK(0227AH)
140 H=PEEK(0227BH)
150 LF=((256*H)+L)/100
160 PRINT "MASS SETPOINT","WEIGHT","ACTUAL MASS","MOTION"
170 PRINT MS,WT,AM,MT
180 PRINT "LIQUID RATIO","LIQUID SET","LIQUID FLOW"
190 PRINT LR,LS,LF
200 Z=PEEK(02285H)
210 IF Z < 128 THEN 230
220 Z=256-Z
225 Z=0-Z
230 L=PEEK(02282H)
231 H=PEEK(02283H)
232 BS=((256*H)+L)*60/200
239 PRINT "STEPPER";Z, "BELT";BS
240 PRINT " "
250 PRINT " "
260 FOR N=0 TO 1000
270 NEXT N
280 GO TO 10

```

Figure 33. Basic Snapshot Program

This frees the master to provide supervisory control and human interface duties.

Certainly, this concept can be expanded to encompass a broad variety of complex control

situations. At the same time, there is no reason why the Intelligent Slave board could not be used to provide a single board solution to a simple control application where no interaction with other processes is required.

IX. CONCLUSION

The purpose of this application note has been to demonstrate some of the techniques which can be used to provide a control system design solution using an intelligent slave concept. This has been done and the system has been constructed and has been found to operate as the design specified. The Intelligent Slave Concept does provide a single board solution to distributed control and certainly offloads the master processor of control duties.

```

10 REM THIS PROGRAM IS USED TO INPUT SETPOINTS
11 REM TO THE LIQUID CONTROL SYSTEM.
20 POK 02387H:0
30 INPUT "ENTER MASS SETPOINT-M8"
40 IF MS > 1500 THEN 20
50 MS=INT(M8*10/60)
60 H=INT(M8/320)
70 L=INT(M8-H*320)
80 POK 02387H:1
90 POK 02387H:1
95 INPUT "PERCENT LIQUID-LR"
100 LR=INT(LR)
110 IF LR > 127 THEN 80
120 POK 02387H:1
130 POK 02387H:1
80 RUN "STATUS"

```

Figure 32. Basic Program for Parameter Initialization

```

PROGRAM NAME:STATUS
10 H=PEEK(0237EH)
20 H=PEEK(0237EH)
30 MS=(1500-H)+L*60/10
40 L=PEEK(0237EH)
50 H=PEEK(0237EH)
60 WT=(1500-H)+L*100
70 L=PEEK(02380H)
80 H=PEEK(02387H)
90 AM=(1500-H)+L*60/10
100 MT=PEEK(02394H)
110 LR=PEEK(02387H)*100
120 LS=AM-LR
130 L=PEEK(0237AH)
140 H=PEEK(0237EH)
150 LR=(1500-H)+L*100
160 PRINT "MASS SETPOINT","WEIGHT","ACTUAL MASS","MOTION"
170 PRINT MS,WT,AM,MT
180 PRINT "LIQUID RATIO","LIQUID SET","LIQUID FLOW"
190 PRINT LR,L,LS
200 Z=PEEK(02382H)
210 IF Z < 128 THEN 230
220 Z=256-Z
230 Z=0-Z
240 L=PEEK(02382H)
250 H=PEEK(02382H)
260 BS=(1500-H)+L*60/200
270 PRINT "STEPPER","Z","BELT",BS
280 PRINT ""
290 PRINT ""
300 FOR N=0 TO 1000
310 NEXT N
320 GO TO 10

```

Figure 33. Basic Snapshot Program

1818-II PLM-88 V3.1 COMPILATION OF MODULE BACKGROUNDMODULE
 OBJECT MODULE PLACED IN: F1:BGKND.08J
 COMPILER INVOKED BY: PLM88: F1:BGKND.PLM DEBUG PAGEWIDTH(72) TITLE('BA
 -CKGROUND PROGRAM')

 * THIS IS THE MAIN BACKGROUND OPERATING *
 * PROGRAM FOR THE PID CONTROL SYSTEM. *

BACKGROUNDMODULE: DO;

/* DECLARATION OF BOARD I/O ASSIGNMENTS */
 DECLARE UP100STATUS LITERALLY '002H';
 DECLARE UP101STATUS LITERALLY '007H';
 DECLARE UP102STATUS LITERALLY '002H';
 DECLARE UP100COMMAND LITERALLY '002H';
 DECLARE UP101COMMAND LITERALLY '007H';
 DECLARE UP102COMMAND LITERALLY '002H';
 DECLARE UP100DATA LITERALLY '000H';
 DECLARE UP101DATA LITERALLY '000H';
 DECLARE UP102DATA LITERALLY '000H';
 DECLARE RESET LITERALLY '000H';

APPENDIX A

/* DECLARATION OF RAM TEST PARAMETERS */
 DECLARE BEGINRAM LITERALLY '0000H';
 DECLARE ENDRAM LITERALLY '0000H';
 DECLARE ZEROSETPATTERN LITERALLY '000H';
 DECLARE ONESETPATTERN LITERALLY '000H';

/* DECLARATION OF RESET LATCH BIT ASSIGNMENTS */
 DECLARE RESETUP100 LITERALLY '00000001H';
 DECLARE RESETUP101 LITERALLY '00000010H';
 DECLARE RESETUP102 LITERALLY '00000100H';
 DECLARE LIGHTLED LITERALLY '00001000H';
 DECLARE MULTISIMTR LITERALLY '00010000H';

/* DECLARATION OF I88C 941 STATUS BITS */
 DECLARE I8F LITERALLY '00000001H';
 DECLARE O8F LITERALLY '00000010H';

/* DECLARATION OF I88C 941 COMMANDS */
 DECLARE IDEN LITERALLY '000H';

/* DECLARATION OF I88C 941 IDENTIFICATION CODE */
 DECLARE 28C941 LITERALLY '41H';

/* DECLARATION OF MEMORY TEST ADDRESS REGISTER */
 DECLARE I ADDRESS AT (87FH);
 DECLARE MEMLOC BASED I BYTE;

/* DECLARATION OF RESET MASKS FOR 88C5 PROCESSOR */

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE BACKGROUNDMODULE
 OBJECT MODULE PLACED IN :F1:BCKGND.OBJ
 COMPILER INVOKED BY: PLM80 :F1:BCKGND.PLM DEBUG PAGEWIDTH(72) TITLE('BA
 -CKGROUND PROGRAM')

```

/*****
* THIS IS THE MAIN BACKGROUND OPERATING
* PROGRAM FOR THE PID CONTROL SYSTEM.
*****/

1      BACKGROUND$MODULE: DO;

/* DECLARATION OF BOARD I/O ASSIGNMENTS */
2      1      DECLARE UPI$0$STATUS      LITERALLY '0E5H';
3      1      DECLARE UPI$1$STATUS      LITERALLY '0E7H';
4      1      DECLARE UPI$2$STATUS      LITERALLY '0E9H';

5      1      DECLARE UPI$0$COMMAND     LITERALLY '0E5H';
6      1      DECLARE UPI$1$COMMAND     LITERALLY '0E7H';
7      1      DECLARE UPI$2$COMMAND     LITERALLY '0E9H';

8      1      DECLARE UPI$0$DATA        LITERALLY '0E4H';
9      1      DECLARE UPI$1$DATA        LITERALLY '0E6H';
10     1      DECLARE UPI$2$DATA        LITERALLY '0E8H';

11     1      DECLARE RESET$LATCH$ADR    LITERALLY '0EAH';

/* DECLARATION OF RAM TEST PARAMETERS */
12     1      DECLARE BEGIN$RAM          LITERALLY '8000H';
13     1      DECLARE END$RAM            LITERALLY '8500H';
14     1      DECLARE ZERO$PATTERN       LITERALLY '000H';
15     1      DECLARE ONE$PATTERN        LITERALLY '0FFH';

/* DECLARATION OF RESET LATCH BIT ASSIGNMENTS */
16     1      DECLARE RESET$UPI$0       LITERALLY '00000001B';
17     1      DECLARE RESET$UPI$1       LITERALLY '00000010B';
18     1      DECLARE RESET$UPI$2       LITERALLY '00000100B';
19     1      DECLARE LIGHT$LED          LITERALLY '00001000B';
20     1      DECLARE MULTI$INTR         LITERALLY '00010000B';

/* DECLARATION OF ISBC 941 STATUS BITS */
21     1      DECLARE IBF                LITERALLY '00000010B';
22     1      DECLARE OBF                LITERALLY '00000001B';

/* DECLARATION OF ISBC 941 COMMANDS */
23     1      DECLARE IDEN               LITERALLY '000H';

/* DECLARATION OF ISBC 941 IDENTIFICATION CODE */
24     1      DECLARE SBC941             LITERALLY '41H';

/* DECLARATION OF MEMORY TEST ADDRESS REGISTER */
25     1      DECLARE I ADDRESS AT (87FEH);
26     1      DECLARE MEMLOC BASED I BYTE;

/* DECLARATION OF RESET MASKS FOR 8085 PROCESSOR */

```

```

27 1 1 DECLARE MASKS BYTE DATA (00FH); /*
/* DECLARATION OF PL/M-80 SIM INSTRUCTION */
28 1 S$MASK: PROCEDURE (MASK) EXTERNAL;
29 2 DECLARE MASK BYTE;
30 2 END S$MASK;

/* DECLARATION OF INITIATION TASK */
31 1 INITIATION:
PROCEDURE EXTERNAL;
32 2 END INITIATION;

/* CLEAR ISBC 941 DEVICES USING ON-BOARD RESET */
33 1 OUTPUT (RESET$LATCH$ADR) = 0;

/* MASK OUT THE RESET INTERRUPTS OF THE PROCESSOR */
34 1 CALL S$MASK (MASKS);

/* TEST MEMORY RAM LOCATIONS */
35 1 DO I = BEGIN$RAM TO END$RAM;
36 2 MEMLOC = ZERO$PATTERN;
37 2 DO WHILE MEMLOC <> ZERO$PATTERN;
38 3 END;

39 2 MEMLOC = ONE$PATTERN;
40 2 DO WHILE MEMLOC <> ONE$PATTERN;
41 3 END;
42 2 END;

/* RELEASE 941 LOCKOUT/RESET BITS */
43 1 OUTPUT (RESET$LATCH$ADR) = RESET$UPI$0 OR
RESET$UPI$1 OR
RESET$UPI$2 OR
MULTI$INTR;

/* VERIFY THAT SBC941 PROCESSOR IS IN SOCKET 0 */
44 1 DO WHILE ((INPUT (UPI$0$STATUS) AND IBF) <> 0);
45 2 END;
46 1 OUTPUT (UPI$0$COMMAND) = IDEN;
47 1 DO WHILE ((INPUT (UPI$0$STATUS) AND OBF) = 0);
48 2 END;
49 1 DO WHILE (INPUT (UPI$0$DATA) <> SBC941);
50 2 END;

/* VERIFY THAT SBC941 PROCESSOR IS IN SOCKET 1 */
51 1 DO WHILE ((INPUT (UPI$1$STATUS) AND IBF) <> 0);
52 2 END;
53 1 OUTPUT (UPI$1$COMMAND) = IDEN;
54 1 DO WHILE ((INPUT (UPI$1$STATUS) AND OBF) = 0);
55 2 END;
56 1 DO WHILE (INPUT (UPI$1$DATA) <> SBC941);
57 2 END;

```



```

58 1      /* VERIFY THAT SBC941 PROCESSOR IS IN SOCKET 2 */
59 2      DO WHILE ((INPUT (UPI$2$STATUS) AND IBF) <> 0);
60 1      END;
61 1      OUTPUT (UPI$2$COMMAND) = IDEN;
62 2      DO WHILE ((INPUT (UPI$2$STATUS) AND OBF) = 0);
63 1      END;
64 2      DO WHILE (INPUT (UPI$2$DATA) <> SBC941);
        END;

        /* START-UP TEST OK- TURN OFF LED */
65 1      OUTPUT (RESET$LATCH$ADR) = RESET$UPI$0 OR
        RESET$UPI$1 OR
        RESET$UPI$2 OR
        LIGHT$LED OR
        MULTI$INTR;

        /* INITIATE THE CONTROL DEVICES */
66 1      CALL INITIATION;

        /* PERFORM BACKGROUND TASKS */
67 1      DO WHILE 1;
68 2      HALT;
69 2      END;

70 1      END BACKGROUND$MODULE;

```

MODULE INFORMATION:

```

CODE AREA SIZE = 00D4H 212D
VARIABLE AREA SIZE = 0000H 0D
MAXIMUM STACK SIZE = 0002H 2D
128 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MAINCONTROLMODULE
 OBJECT MODULE PLACED IN :F1:CNTTSK.OBJ
 COMPILER INVOKED BY: PLM80 :F1:CNTTSK.PLM DEBUG

```

$INTVECTOR(4,3F00H)
$PAGEWIDTH(72)
$TITLE('MAIN CONTROL')
/*****
*
*      MAIN$CONTROL$TASK
* THIS TASK IS USED TO CONTROL THE TWO PID CONTROL
* LOOPS. ONE LOOP CONTROLS THE SPEED OF A CONVEYOR
* WHILE THE SECOND CONTROLS THE FLOW OF A LIQUID.
* THE TASK OPERATES EACH 200 MSEC.
*
* ***** VERSION 1.1 *****/
1  MAIN$CONTROL$MODULE: DO;
  /* DECLARATION OF PID RECORD SET-UP TASK */
2  1  UQPSSET:
      PROCEDURE (PR$PTR,ERROR$FLD$PTR,PRIV$PTR) EXTERNAL
      ;
3  2  DECLARE (PR$PTR,ERROR$FLD$PTR,PRIV$PTR) ADDRESS;
4  2  END UQPSSET;

  /* DECLARATION OF PID CONTROL BITS */
5  1  UQPSCT:
      PROCEDURE (PR$PTR,CONTROL$PTR) EXTERNAL;
6  2  DECLARE (PR$PTR,CONTROL$PTR) ADDRESS;
7  2  END UQPSCT;

  /* PROCEDURE TO SET UP PID CONSTANTS */
8  1  UQPSCN:
      PROCEDURE (PR$PTR,CONSTANT$PTR) EXTERNAL;
9  2  DECLARE (PR$PTR,CONSTANT$PTR) ADDRESS;
10 2  END UQPSCN;

  /* DEFINE THE DEFAULT ERROR HANDLER */
11 1  UQPSBD:
      PROCEDURE (PR$PTR,BOUND$PTR) EXTERNAL;
12 2  DECLARE (PR$PTR,BOUND$PTR) ADDRESS;
13 2  END UQPSBD;

  /* PROCEDURE TO CHANGE THE TIME INTERVAL */
14 1  UQPSTI:
      PROCEDURE (PR$PTR,TIME$INTERVAL$PTR) EXTERNAL;
15 2  DECLARE (PR$PTR,TIME$INTERVAL$PTR) ADDRESS;
16 2  END UQPSTI;

  /* DECLARATION OF THE PID CONTROL PROGRAM */
17 1  UQPPID:
      PROCEDURE (PR$PTR,IR$PTR) EXTERNAL;
18 2  DECLARE (PR$PTR,IR$PTR) ADDRESS;
19 2  END UQPPID;

```

```

20 1      /* DECLARATION OF WEIGHBELT SPEED INTERFACE */
      WEIGHBELT$SPEED:
21 2          PROCEDURE BYTE EXTERNAL;
          END WEIGHBELT$SPEED;

      /* DECLARATION OF WEIGHBELT WEIGHT INTERFACE */
22 1      WEIGHBELT$WEIGHT:
      PROCEDURE ADDRESS EXTERNAL;
23 2          END WEIGHBELT$WEIGHT;

      /* DECLARATION OF LIQUID FLOW RATE INTERFACE */
24 1      LIQUID$FLOW$RATE:
      PROCEDURE ADDRESS EXTERNAL;
25 2          END LIQUID$FLOW$RATE;

      /* DECLARATION OF WEIGHBELT MOTOR DRIVE INTERFACE */
26 1      WEIGHBELT$MOTOR$DRIVE:
      PROCEDURE (SPEED) EXTERNAL;
27 2          DECLARE SPEED ADDRESS;
28 2          END WEIGHBELT$MOTOR$DRIVE;

      /* DECLARATION OF LIQUID VALVE INTERFACE */
29 1      LIQUID$VALVE$POSITION:
      PROCEDURE (POSITION) EXTERNAL;
30 2          DECLARE POSITION BYTE;
31 2          END LIQUID$VALVE$POSITION;

      /* DECLARATION OF PROCESSOR 0 INITIALIZATION MODULE */
32 1      PROCESSOR$0$INITIALIZATION:
      PROCEDURE EXTERNAL;
33 2          END PROCESSOR$0$INITIALIZATION;

      /* DECLARATION OF PROCESSOR 1 INITIALIZATION MODULE */
34 1      PROCESSOR$1$INITIALIZATION:
      PROCEDURE EXTERNAL;
35 2          END PROCESSOR$1$INITIALIZATION;

      /* DECLARATION OF PROCESSOR 2 INITIALIZATION MODULE */
36 1      PROCESSOR$2$INITIALIZATION:
      PROCEDURE EXTERNAL;
37 2          END PROCESSOR$2$INITIALIZATION;

      /* DECLARATION OF PIT COUNTER 1 INITIALIZATION */
38 1      COUNTER$1$INITIALIZATION:
      PROCEDURE EXTERNAL;
39 2          END COUNTER$1$INITIALIZATION;

      /* DECLARATION OF PIT COUNTER 2 INITIALIZATION */
40 1      COUNTER$2$INITIALIZATION:
      PROCEDURE EXTERNAL;
41 2          END COUNTER$2$INITIALIZATION;

```

```

42 1      /* DECLARATION OF FSP UNSIGNED LOAD PROCEDURES */
43 2      MQULD1: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
44 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
45 1      MQULD2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
46 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
47 2      END MQULD2;

48 1      /* DECLARATION OF FSP UNSIGNED MULTIPLY PROCEDURE */
49 2      MQUML1: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
50 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
51 1      MQUML2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
52 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
53 2      END MQUML2;

54 1      /* DECLARATION OF FSP UNSIGNED DIVIDE PROCEDURE */
55 2      MQUDV1: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
56 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
57 1      MQUDV2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
58 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
59 2      END MQUDV2;

60 1      /* DECLARATION OF FSP SIGNED DIVIDE PROCEDURE */
61 2      MQSDV1: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
62 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
63 1      MQSDV2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
64 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
65 2      END MQSDV2;

66 1      /* DECLARATION OF FSP SIGNED STORE PROCEDURE */
67 2      MQSST2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
68 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
69 1      /* DECLARATION OF FSP SIGNED LOAD PROCEDURE */
70 2      MQSLD2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
71 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
72 1      /* DECLARATION OF FSP SIGNED SUBTRACT PROCEDURE */
73 2      MQSSB2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
74 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
75 1      /* DECLARATION OF FSP UNSIGNED STORE PROCEDURE */
76 2      MQUST1: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
77 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
78 1      MQUST2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
79 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
80 2      END MQUST2;

```



```

/* DECLARATION OF FSP SIGNED MULTIPLY PROCEDURE */
81 1 MQSML1: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
82 2 DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
83 2 END MQSML1;

$EJECT
/*****
* DATA STORAGE AREAS FOR THE PID CONTROL *
*****/

/* DEFINITION OF LIMITATION CONSTANTS */
84 1 DECLARE MAX$MOTOR$SPEED LITERALLY '550';
85 1 DECLARE MIN$MOTOR$SPEED LITERALLY '0';
86 1 DECLARE MAX$VALVE$MOVEMENT LITERALLY '10';
87 1 DECLARE MIN$VALVE$MOVEMENT LITERALLY '-10';

/* DEFINITION OF PID PARAMETER COEFFICIENTS */
88 1 DECLARE FEEDER$C0 LITERALLY '1';
89 1 DECLARE FEEDER$C1 LITERALLY '1';
90 1 DECLARE FEEDER$C2 LITERALLY '1';
91 1 DECLARE FEEDER$C3 LITERALLY '1';
92 1 DECLARE FEEDER$TIME$INTERVAL LITERALLY '1';
93 1 DECLARE FEEDER$SCALE$FACTOR LITERALLY '1';

94 1 DECLARE LIQUID$C0 LITERALLY '1';
95 1 DECLARE LIQUID$C1 LITERALLY '1';
96 1 DECLARE LIQUID$C2 LITERALLY '1';
97 1 DECLARE LIQUID$C3 LITERALLY '1';
98 1 DECLARE LIQUID$TIME$INTERVAL LITERALLY '1';
99 1 DECLARE LIQUID$SCALE$FACTOR LITERALLY '10';

/* DEFINITION OF RESET LATCH PARAMETERS */
100 1 DECLARE RESET$LATCH$ADR LITERALLY '0EAH';
101 1 DECLARE INDICATOR$ON LITERALLY '07H';
102 1 DECLARE INDICATOR$OFF LITERALLY '0FH';

/* RESERVE 18 BYTES FOR THE INTEGER RECORD */
103 1 DECLARE IR (18) BYTE PUBLIC;

/* RESERVE 42 BYTES FOR EACH PID RECORD */
104 1 DECLARE PRCV (42) BYTE;
105 1 DECLARE PRLQ (42) BYTE;

/* RESERVE SPACE FOR COUNTER DATA */
106 1 DECLARE (LIQ$COUNT,BELT$COUNT) BYTE PUBLIC;

/* RESERVE 12 BYTES FOR EACH CONSTANT ARRAY */
107 1 DECLARE CONSTANTS1 STRUCTURE (
C0 ADDRESS,
C1 ADDRESS,
C2 ADDRESS,
C3 ADDRESS,
DT ADDRESS,
S ADDRESS );

```

```

108      1      DECLARE CONSTANTS2 STRUCTURE (
                C0 ADDRESS,
                C1 ADDRESS,
                C2 ADDRESS,
                C3 ADDRESS,
                DT ADDRESS,
                S ADDRESS );
/* RESERVE 8 BYTES FOR EACH BOUNDS ARRAY */
109      1      DECLARE BOUNDS1 (4) ADDRESS DATA (
                000H,
                000H,
                MAX$MOTOR$SPEED,
                MIN$MOTOR$SPEED );
110      1      DECLARE BOUNDS2 (4) ADDRESS DATA (
                000D,
                000D,
                MAX$VALVE$MOVEMENT,
                MIN$VALVE$MOVEMENT );
/* RESERVE 1 BYTE FOR EACH CONTROL BYTE */
111      1      DECLARE CONTROL1 BYTE DATA (073H);
112      1      DECLARE CONTROL2 BYTE DATA (053H);
/* DECLARE TIME INTERVAL */
113      1      DECLARE TIME$INTERVAL ADDRESS DATA (1);
/* RESERVE SPACE FOR THE CURRENT BELT SPEED */
114      1      DECLARE BELT$SPEED BYTE;
/* RESERVE SPACE FOR THE CURRENT BELT WEIGHT */
115      1      DECLARE BELT$WEIGHT ADDRESS;
/* RESERVE SPACE FOR THE LIQUID FLOW */
116      1      DECLARE LIQUID$FLOW ADDRESS;
/* RESERVE SPACE FOR THE EFFECTIVE SETPOINT */
117      1      DECLARE MASS$SETPOINT ADDRESS;
/* RESERVE SPACE FOR THE DESIRED SETPOINT */
118      1      DECLARE SET$POINT ADDRESS;
/* RESERVE SPACE FOR THE DISTANCE OF BELT PER REVOLUTION */
119      1      DECLARE DIST$REV BYTE DATA (100);
/* DEFINE THE CONVEYOR LENGTH */
120      1      DECLARE CONV$LENGTH BYTE DATA (200);
/* DEFINE THE CONSTANT SIX */
121      1      DECLARE SIX BYTE DATA (6);
/* RESERVE STORAGE FOR ACTUAL CURRENT MASS FLOW */
122      1      DECLARE MASS$FLOW ADDRESS;

```

```

123 1 /* RESERVE SPACE FOR BELT CONTROL OUTPUT */
      DECLARE BELT$CONTROL ADDRESS;

124 1 /* RESERVE SPACE FOR LIQUID RATIO */
      DECLARE LIQUID$RATIO BYTE;

125 1 /* RESERVE SPACE FOR LIQUID CONTROL OUTPUT */
      DECLARE LIQUID$VALVE ADDRESS;

126 1 /* RESERVE SPACE FOR RUN/HAUT CONTROL */
      DECLARE SYSTEM$RUNNING BYTE PUBLIC;

127 1 /* RESERVE SPACE FOR ERROR FIELD */
128 1 DECLARE ERROR$FIELD ADDRESS DATA (0F800H);
      DECLARE DUMMY ADDRESS;

129 1 /* RESERVE SPACE FOR PIC ICW BYTE */
      DECLARE ICW BYTE;

130 1 /* DEFINE CONSTANT 1000 */
      DECLARE THOUSAND ADDRESS DATA (1000);

131 1 /* DEFINE CONSTANT 0 */
      DECLARE ZERO ADDRESS DATA (0);

132 1 /* DEFINE INTERRUPT JUMP TABLE */
      DECLARE JUMP$TABLE BYTE AT (3F00H);

133 1 /* DECLARATION OF PIC ADDRESSES ON ISBC 569 BOARD */
134 1 DECLARE PIC$ICW1$PTR LITERALLY '0ECH';
135 1 DECLARE PIC$ICW2$PTR LITERALLY '0EDH';
      DECLARE PIC$INT$MASK$PTR LITERALLY '0EDH';

136 1 /* DECLARATION OF PIC CONSTANTS */
137 1 DECLARE CLR$LOW$BITS LITERALLY '0ECH';
138 1 DECLARE INTERVAL$4 LITERALLY '016H';
      DECLARE INTERRUPT$MASK LITERALLY '0F4H';

      $EJECT
      /*****
      * INITIALIZE PROGRAM AT START-UP OF SYSTEM *
      * THIS PROCEDURE IS CALLED AT START-UP *
      *****/

139 1 INITIATION: PROCEDURE PUBLIC;

140 2 /* DISABLE THE INTERRUPTS */
      DISABLE;

141 2 /* INITIALIZE PID RECORD */
142 2 CALL UQPSET (.PRCV,.ERROR$FIELD,.DUMMY);
      CALL UQPSET (.PRLO,.ERROR$FIELD,.DUMMY);

```

```

/* INITIALIZE THE CONTROL BITS */
143 2 CALL UQPSCT (.PRCV,.CONTROL1);
144 2 CALL UQPSCT (.PRLQ,.CONTROL2);

/* SET UP THE PID CONSTANTS */
145 2 CONSTANTS1.C0 = FEEDER$C0;
146 2 CONSTANTS1.C1 = FEEDER$C1;
147 2 CONSTANTS1.C2 = FEEDER$C2;
148 2 CONSTANTS1.C3 = FEEDER$C3;
149 2 CONSTANTS1.DT = FEEDER$TIME$INTERVAL;
150 2 CONSTANTS1.S = FEEDER$SCALE$FACTOR;

151 2 CONSTANTS2.C0 = LIQUID$C0;
152 2 CONSTANTS2.C1 = LIQUID$C1;
153 2 CONSTANTS2.C2 = LIQUID$C2;
154 2 CONSTANTS2.C3 = LIQUID$C3;
155 2 CONSTANTS2.DT = LIQUID$TIME$INTERVAL;
156 2 CONSTANTS2.S = LIQUID$SCALE$FACTOR;

/* CLEAR SETPOINTS */
157 2 SETPOINT = 0;
158 2 LIQUID$RATIO = 0;
159 2 SYSTEM$RUNNING = 0;

/* INITIALIZE THE CONSTANTS */
160 2 CALL UQPSCT (.PRCV,.CONSTANTS1);
161 2 CALL UQPSCT (.PRLQ,.CONSTANTS2);

/* INITIALIZE THE BOUNDS */
162 2 CALL UQPSBD (.PRCV,.BOUNDS1);
163 2 CALL UQPSBD (.PRLQ,.BOUNDS2);

/* SET THE TIME INTERVAL */
164 2 CALL UQPSTI (.PRCV,.TIME$INTERVAL);
165 2 CALL UQPSTI (.PRLQ,.TIME$INTERVAL);

/* INITIALIZE PROCESSOR 0 */
166 2 CALL PROCESSOR$0$INITIALIZATION;

/* INITIALIZE PROCESSOR 1 */
167 2 CALL PROCESSOR$1$INITIALIZATION;

/* INITIALIZE PROCESSOR 2 */
168 2 CALL PROCESSOR$2$INITIALIZATION;

/* INITIALIZE COUNTER 1 */
169 2 CALL COUNTER$1$INITIALIZATION;

/* INITIALIZE COUNTER 2 */
170 2 CALL COUNTER$2$INITIALIZATION;

/* INITIALIZE INTERRUPT CONTROLLER */
171 2 ICW = (LOW (.JUMP$TABLE) AND
          CLR$LOW$BITS ) OR
          INTERVAL$4 ;
172 2 OUTPUT (PIC$ICW1$PTR) = ICW;

```



```

173 2      ICW = HIGH (.JUMP$TABLE);
174 2      OUTPUT (PIC$ICW2$PTR) = ICW;

/* SET INTERRUPT MASKS */
175 2      OUTPUT (PIC$INT$MASK$PTR) = INTERRUPT$MASK;

/* ENABLE INTERRUPTS */
176 2      ENABLE;

/* RETURN TO MAIN PROGRAM */
177 2      RETURN;

178 2      END INITIATION;
$EJECT
/*****
* THIS IS THE PID CONTROL ROUTINE. IT IS ENTERED *
* EACH 200 MILLISECONDS THROUGH AN INTERRUPT GEN- *
* ERATED BY THE FREQUENCY COUNTER UPI AND SENT TO *
* INTERRUPT 3. *
*****/

179 1      PIDRUN:  PROCEDURE INTERRUPT 3 PUBLIC;

/* TURN THE LED INDICATOR ON */
180 2      OUTPUT (RESET$LATCH$ADR) = INDICATOR$ON;

/* GET WEIGHBELT WEIGHT */
181 2      BELT$WEIGHT=WEIGHBELT$WEIGHT;

/* GET LIQUID FLOW RATE */
182 2      LIQUID$FLOW=LIQUID$FLOW$RATE;

/* CONTROL START-STOP RAMP */
183 2      IF SYSTEM$RUNNING
185 2      THEN MASS$SETPOINT=SETPOINT;
      ELSE MASS$SETPOINT=0;

/* DETERMINE ACTUAL MASS FLOW ON WEIGHBELT */
186 2      CALL MQULD2(.IR,.BELT$CONTROL);
187 2      CALL MQUML2(.IR,.BELT$WEIGHT);
188 2      CALL MQUML1(.IR,.DIST$REV);
189 2      CALL MQUDV1(.IR,.CONV$LENGTH);
190 2      CALL MQSDV2(.IR,.THOUSAND);
191 2      CALL MQSST2(.IR,.MASS$FLOW);

/* COMPUTE ERROR SIGNAL ON WEIGHBELT */
192 2      CALL MQSLD2(.IR,.MASS$SETPOINT);
193 2      CALL MQSSB2(.IR,.MASS$FLOW);

/* HANDLE PID BELT CONTROL ALGORITHM */
194 2      CALL UQPPID(.PRCV,.IR);

/* STORE OUTPUT SIGNAL */
195 2      CALL MQUST2(.IR,.BELT$CONTROL);

```

```

196 2      /* COMPUTE LIQUID SETPOINT */
197 2      CALL MQSLD2(.IR,.MASS$FLOW);
198 2      CALL MQSML1(.IR,.LIQUID$RATIO);
      CALL MQSML1(.IR,.SIX);

      /* VERIFY THAT WEIGHBELT IS MOVING */
199 2      IF WEIGHBELT$SPEED = 0
      THEN CALL MQULD2(.IR,.ZERO);

201 2      /* COMPUTE LIQUID ERROR */
      CALL MQSSB2(.IR,.LIQUID$FLOW);

      /* HANDLE PID LIQUID CONTROL */
202 2      CALL UQPPID(.PRLQ,.IR);

      /* STORE OUTPUT SIGNAL */
203 2      CALL MQUST1(.IR,.LIQUID$VALVE);

      /* OUTPUT WEIGHBELT CONTROL SIGNAL */
204 2      CALL WEIGHBELT$MOTOR$DRIVE (BELT$CONTROL);

      /* OUTPUT FLOW CONTROL SIGNAL */
205 2      CALL LIQUID$VALVE$POSITION (LIQUID$VALVE);

      /* SEND END OF INTERRUPT TO 8259 CONTROLLER */
206 2      OUTPUT(0ECH)=020H;

      /* TURN THE LED INDICATOR OFF */
207 2      OUTPUT (RESET$LATCH$ADR) = INDICATOR$OFF;

      /* RETURN FROM CONTROL TASK */
208 2      RETURN;
209 2      END PIDRUN;
210 1      END;

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 01C1H      449D
VARIABLE AREA SIZE = 0094H      148D
MAXIMUM STACK SIZE = 000AH      10D
465 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PROCESSORINITIALIZATIONMODULE
 OBJECT MODULE PLACED IN :F1:SBC941.OBJ
 COMPILER INVOKED BY: PLM80:F1:SBC941.PLM DEBUG PAGEWIDTH(72) TITLE('PR
 -OCESSOR INITIALIZATION')

```

/*****
* THIS PROGRAM IS USED TO INITIALIZE THE ISBC *
* 941 PROCESSOR INSTALLED IN SOCKET 0. THE *
* DEVICE WILL OPERATE IN THE FREQUENCY OUTPUT *
* MODE. *
*****/

1      PROCESSOR$INITIALIZATION$MODULE: DO;

      /* DECLARATION OF ADDRESSES */
2      1      DECLARE UPI$0$STATUS      LITERALLY '0E5H';
3      1      DECLARE UPI$0$COMMAND    LITERALLY '0E5H';
4      1      DECLARE UPI$0$DATA       LITERALLY '0E4H';

5      1      DECLARE UPI$1$STATUS      LITERALLY '0E7H';
6      1      DECLARE UPI$1$COMMAND    LITERALLY '0E7H';
7      1      DECLARE UPI$1$DATA       LITERALLY '0E6H';

8      1      DECLARE UPI$2$STATUS      LITERALLY '0E9H';
9      1      DECLARE UPI$2$COMMAND    LITERALLY '0E9H';
10     1      DECLARE UPI$2$DATA       LITERALLY '0E8H';

      /* DECLARATION OF ISBC 941 COMMANDS */
11     1      DECLARE SETP1             LITERALLY '00BH';
12     1      DECLARE CLRP1             LITERALLY '00DH';
13     1      DECLARE CLRP2             LITERALLY '00EH';
14     1      DECLARE PAUSE             LITERALLY '005H';
15     1      DECLARE LOOP              LITERALLY '004H';
16     1      DECLARE INITPF            LITERALLY '002H';
17     1      DECLARE PACIFY            LITERALLY '001H';
18     1      DECLARE ENFLAG            LITERALLY '006H';

      /* DECLARATION OF ISBC 941 STATUS BITS */
19     1      DECLARE RFC               LITERALLY '080H';
20     1      DECLARE IBF               LITERALLY '002H';
21     1      DECLARE QF                LITERALLY '010H';

      /* DECLARATION OF ISBC 941 #0 INITIALIZATION DATA */
22     1      DECLARE FREQ              LITERALLY '0B5H';
23     1      DECLARE SF                LITERALLY '000H';
24     1      DECLARE OUTPUT$ENABLE0    LITERALLY '001H';
25     1      DECLARE INITIAL$STATE     LITERALLY '000H';
26     1      DECLARE DELAY             LITERALLY '001H';
27     1      DECLARE PERIOD            LITERALLY '000H';
28     1      DECLARE INITIAL$OUTPUT    LITERALLY '00EH';

```

```

29 1      /* DECLARATION OF INTERVAL TIMER PARAMETERS */
30 1      DECLARE PIT$0$MODE LITERALLY '016H';
31 1      DECLARE PIT$0$INTERVAL LITERALLY '00EH';
32 1      DECLARE PIT$0$MODE$WRD LITERALLY '0E3H';
33 1      DECLARE PIT$0$COUNT LITERALLY '0E0H';

33 1      /* DECLARATION OF COUNTER LOCATIONS */
34 1      DECLARE (LIQ$COUNT,BELT$COUNT) BYTE EXTERNAL;

34 1      /* DECLARATION OF ISBC 941 PRIMARY DATA */
35 1      DECLARE INIT$0$TABLE(6) BYTE DATA (
        FREQ,
        SF,
        OUTPUT$ENABLE0,
        INITIAL$STATE,
        DELAY,
        PERIOD );

35 1      /* DECLARATION OF MISC PARAMETERS */
36 1      DECLARE I BYTE;

36 1      /******
37 1      *      INITIALIZATION PROGRAM BODY      *
38 1      *      ******/

36 1      PROCESSOR$0$INITIALIZATION: PROCEDURE PUBLIC;

37 2      /* INITIALIZE COUNTER 0 FOR 10 MICROSECONDS */
38 2      OUTPUT(PIT$0$MODE$WRD)=PIT$0$MODE;
39 2      OUTPUT(PIT$0$COUNT)=PIT$0$INTERVAL;

39 2      /* VERIFY THAT PROCESSOR IS RESET */
40 3      DO WHILE ((INPUT(UPI$0$STATUS) AND RFC) = 0);
41 4      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
42 3      END;
43 3      OUTPUT(UPI$0$COMMAND)=PACIFY;
44 2      END;

44 2      /* REQUEST PRIMARY FUNCTION */
45 3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
46 2      END;
47 2      OUTPUT(UPI$0$COMMAND)= INITPF;

47 2      /* LOAD INITIAL PARAMETERS */
48 3      DO I=0 TO 5;
49 4      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
50 3      END;
51 3      OUTPUT(UPI$0$DATA)=INIT$0$TABLE(I);
52 2      END;

52 2      /* TERMINATE PARAMETER LOADING */
53 3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
54 2      END;
55 2      OUTPUT(UPI$0$COMMAND)=PAUSE;

```



```

56 3      END;
57 2      OUTPUT(UPI$0$COMMAND)=LOOP;

/* SET UNUSED BITS TO ALLOW EXPANSION */

58 2      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
59 3      END;
60 2      OUTPUT(UPI$0$COMMAND)=CLRP2;

61 2      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
62 3      END;
63 2      OUTPUT(UPI$0$DATA)=INITIAL$OUTPUT;

/* RETURN TO CALLING PROGRAM */
64 2      RETURN;

65 2      END PROCESSOR$0$INITIALIZATION;
$EJECT
/*****
* THIS PROCEDURE IS USED TO INITIALIZE THE ISBC *
* 941 PROCESSOR INSTALLED IN SOCKET 1. THE DE- *
* VICE WILL OPERATE IN THE FCOUNT, HIGH FRE- *
* QUENCY INPUT MODE. *
*****/

/* DEFINE INITIALIZATION PARAMETERS */
66 1      DECLARE FCOUNT      LITERALLY '033H';
67 1      DECLARE INPUT$SELECT LITERALLY '001H';
68 1      DECLARE OUTPUT$ENABLE$1 LITERALLY '001H';
69 1      DECLARE SAMPLING$INTERVAL LITERALLY '009H';
70 1      DECLARE INITIAL$STATE$1 LITERALLY '0E1H';

/* DECLARE PARAMETER INITIALIZATION TABLE */
71 1      DECLARE INIT$1$TABLE(4) BYTE DATA (
          FCOUNT,
          INPUT$SELECT,
          OUTPUT$ENABLE$1,
          SAMPLING$INTERVAL );

/*****
*      INITIALIZATION BODY
*****/

72 1      PROCESSOR$1$INITIALIZATION: PROCEDURE PUBLIC;

/* VERIFY THAT PROCESSOR IS RESET */
73 2      DO WHILE ((INPUT(UPI$1$STATUS) AND RFC) = 0);
74 3      DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
75 4      END;
76 3      OUTPUT(UPI$1$COMMAND)=PACIFY;
77 3      END;

```

```

78 2 2000' /* REQUEST PRIMARY FUNCTION */
79 3 3000' DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
80 2 2500' END;
      OUTPUT(UPI$1$COMMAND)=INITPF;

      /* LOAD INITIAL PARAMETERS */
81 2 2001' DO I=0 TO 3;
82 3 3001' DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
83 4 4001' END;
84 3 3002' OUTPUT(UPI$1$DATA)=INIT$1$TABLE(I);
85 3 3003' END;

      /* TERMINATE PARAMETER LOADING */
86 2 2004' DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
87 3 3004' END;
88 2 2005' OUTPUT(UPI$1$COMMAND)=PAUSE;

      /* SET UNUSED BITS HIGH FOR SPARE ENABLES */
89 2 2006' DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
90 3 3006' END;
91 2 2007' OUTPUT(UPI$1$COMMAND)=SETPI;
92 2 2008' DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
93 3 3008' END;
94 2 2009' OUTPUT(UPI$1$DATA)=INITIAL$STATE$1;

      /* START FREQUENCY COUNT OPERATION */
95 2 2010' DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
96 3 3010' END;
97 2 2011' OUTPUT(UPI$1$COMMAND)=LOOP;

      /* RETURN TO CALLING PROGRAM */
98 2 2012' RETURN;

99 2 2013' END PROCESSOR$1$INITIALIZATION;

$SELECT
/*****
* THIS PROCEDURE IS USED TO INITIALIZE THE ISBC *
* 941 INSTALLED IN SOCKET 2. THE DEVICE WILL BE *
* OPERATED AS A STEPPER MOTOR DRIVER. *****/

      /* DEFINE INITIALIZATION PARAMETERS */
100 1 1000' DECLARE STEPPER LITERALLY '017H';
101 1 1001' DECLARE SCALE$FACTOR LITERALLY '0DFH';
102 1 1002' DECLARE OUTPUT$ENABLE$2 LITERALLY '0F0H';
103 1 1003' DECLARE OUTPUT$POLARITY LITERALLY '050H';
104 1 1004' DECLARE COMMON$PERIOD LITERALLY '004H';
105 1 1005' DECLARE P20$STRAN1 LITERALLY '000H';
106 1 1006' DECLARE P20$STRAN2 LITERALLY '000H';
107 1 1007' DECLARE P21$STRAN1 LITERALLY '000H';
108 1 1008' DECLARE P21$STRAN2 LITERALLY '000H';
109 1 1009' DECLARE P22$STRAN1 LITERALLY '000H';
110 1 1010' DECLARE P22$STRAN2 LITERALLY '000H';

```

```

111 1 DECLARE P23$STRAN1 LITERALLY '000H';
112 1 DECLARE P23$STRAN2 LITERALLY '000H';
113 1 DECLARE P24$STRAN1 LITERALLY '000H';
114 1 DECLARE P24$STRAN2 LITERALLY '002H';
115 1 DECLARE P25$STRAN1 LITERALLY '000H';
116 1 DECLARE P25$STRAN2 LITERALLY '002H';
117 1 DECLARE P26$STRAN1 LITERALLY '001H';
118 1 DECLARE P26$STRAN2 LITERALLY '003H';
119 1 DECLARE P27$STRAN1 LITERALLY '001H';
120 1 DECLARE P27$STRAN2 LITERALLY '003H';

121 1 DECLARE CLR$LOW$PORT LITERALLY '00FH';

/* DECLARE PARAMETER INITIALIZATION TABLE */
122 1 DECLARE INIT$2$TABLE(21) BYTE DATA (
    STEPPER,
    SCALE$FACTOR,
    OUTPUT$ENABLE$2,
    OUTPUT$POLARITY,
    COMMON$PERIOD,
    P20$STRAN1,
    P20$STRAN2,
    P21$STRAN1,
    P21$STRAN2,
    P22$STRAN1,
    P22$STRAN2,
    P23$STRAN1,
    P23$STRAN2,
    P24$STRAN1,
    P24$STRAN2,
    P25$STRAN1,
    P25$STRAN2,
    P26$STRAN1,
    P26$STRAN2,
    P27$STRAN1,
    P27$STRAN2
);
/*****
* INITIALIZATION BODY
*****/

123 1 PROCESSOR$2$INITIALIZATION: PROCEDURE PUBLIC;

/* VERIFY THAT PROCESSOR IS RESET */
124 2 DO WHILE ((INPUT(UPI$2$STATUS) AND RFC) = 0);
125 3 DO WHILE ((INPUT(UPI$2$STATUS) AND IBF) <> 0);
126 4 END;
127 3 OUTPUT(UPI$2$COMMAND)=PACIFY;
128 3 END;

/* REQUEST PRIMARY FUNCTION */
129 2 DO WHILE ((INPUT(UPI$2$STATUS) AND IBF) <> 0);
130 3 END;
131 2 OUTPUT(UPI$2$COMMAND)=INITPF;

```

```

132      2      /* LOAD INITIAL PARAMETERS */
133      3          DO I=0 TO 20;
134      4              DO WHILE ((INPUT(UPI$2$STATUS) AND IBF) <> 0);
135      3                  END;
136      3                  OUTPUT(UPI$2$DATA)=INIT$2$TABLE(I);
137      3              END;
138      3          /* TERMINATE PARAMETER LOADING */
139      2          DO WHILE ((INPUT(UPI$2$STATUS) AND IBF) <> 0);
140      3              END;
141      2              OUTPUT(UPI$2$COMMAND)=PAUSE;
142      2          /* START STEPPER CONTROLLER OPERATION */
143      2          DO WHILE ((INPUT(UPI$2$STATUS) AND IBF) <> 0);
144      3              END;
145      2              OUTPUT(UPI$2$COMMAND)=CLRPI;
146      2          DO WHILE ((INPUT(UPI$2$STATUS) AND IBF) <> 0);
147      3              END;
148      2              OUTPUT(UPI$2$DATA)=CLR$LOW$PORT;
149      2          /* RETURN TO CALLING PROGRAM */
150      2          RETURN;
151      2          END PROCESSOR$2$INITIALIZATION;
152      2          $EJECT
153      2          /* *****
154      2          * THIS PROCEDURE IS USED TO INITIALIZE COUNTER *
155      2          * 1 TO PERFORM AS AN EIGHT BIT BINARY COUNTER *
156      2          * WHICH WILL BE USED TO MEASURE THE BELT SPEED.*
157      2          * ***** */
158      2          COUNTER$1$INITIALIZATION: PROCEDURE PUBLIC;
159      2          /* INITIALIZE COUNTER 1 FOR EIGHT BIT COUNTING */
160      2          LIQ$COUNT = 0;
161      2          /* RETURN TO CALLING PROGRAM */
162      2          RETURN;
163      2          END COUNTER$1$INITIALIZATION;
164      2          $EJECT
165      2          /* *****
166      2          * THIS PROCEDURE IS USED TO INITIALIZE COUNTER *
167      2          * 2 TO PERFORM AS AN EIGHT BIT BINARY COUNTER *
168      2          * WHICH WILL BE USED TO MEASURE THE LIQUID *
169      2          * FLOW THROUGH THE METER. *
170      2          * ***** */

```



```

155 1 COUNTER$2$INITIALIZATION: PROCEDURE PUBLIC;
/* INITIALIZE COUNTER 2 FOR EIGHT BIT COUNTING */
156 2 BELT$COUNT = 0;

/* RETURN TO CALLING PROGRAM */
157 2 RETURN;
158 2 END COUNTER$2$INITIALIZATION;
159 1 END PROCESSOR$INITIALIZATION$MODULE;
$EJECT

```

MODULE INFORMATION:

```

CODE AREA SIZE = 0201H 513D
VARIABLE AREA SIZE = 0001H 1D
MAXIMUM STACK SIZE = 0000H 0D
329 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PROCESSORINTERFACEMODULE
 OBJECT MODULE PLACED IN :F1:OPR941.OBJ
 COMPILER INVOKED BY: PLM80 :F1:OPR941.PLM DEBUG

```

$INTVECTOR(4,3F00H)
$PAGEWIDTH(72)
$TITLE('PROCESSOR INTERFACE')
/*****
* THESE PROGRAMS PROVIDE THE OPERATING INTER-
* FACE BETWEEN THE APPLICATION PROGRAM AND
* THE ISBC 941 PROCESSORS OR COUNTERS.
*****/

1      PROCESSOR$INTERFACE$MODULE: DO;

      /* DECLARATION OF ADDRESSES */
2      1      DECLARE UPI$0$STATUS      LITERALLY '0E5H';
3      1      DECLARE UPI$0$COMMAND    LITERALLY '0E5H';
4      1      DECLARE UPI$0$DATA        LITERALLY '0E4H';

5      1      DECLARE UPI$1$STATUS      LITERALLY '0E7H';
6      1      DECLARE UPI$1$COMMAND    LITERALLY '0E7H';
7      1      DECLARE UPI$1$DATA        LITERALLY '0E6H';

8      1      DECLARE UPI$2$STATUS      LITERALLY '0E9H';
9      1      DECLARE UPI$2$COMMAND    LITERALLY '0E9H';
10     1      DECLARE UPI$2$DATA        LITERALLY '0E8H';

      /* DECLARATION OF ISBC 941 COMMANDS */
11     1      DECLARE SETP1              LITERALLY '00BH';
12     1      DECLARE CLRP1              LITERALLY '00DH';
13     1      DECLARE CLRP2              LITERALLY '00EH';
14     1      DECLARE RDFC0              LITERALLY '042H';
15     1      DECLARE RDFC1              LITERALLY '043H';
16     1      DECLARE PAUSE              LITERALLY '005H';
17     1      DECLARE LOOP               LITERALLY '004H';
18     1      DECLARE INITPF             LITERALLY '002H';
19     1      DECLARE PACIFY             LITERALLY '001H';
20     1      DECLARE ENFLAG             LITERALLY '006H';
21     1      DECLARE SETOE              LITERALLY '071H';

      /* DECLARATION OF ISBC 941 STATUS BITS */
22     1      DECLARE RFC                LITERALLY '080H';
23     1      DECLARE IBF                LITERALLY '002H';
24     1      DECLARE OBF                LITERALLY '001H';
25     1      DECLARE QF                 LITERALLY '010H';
26     1      DECLARE QNE                LITERALLY '020H';

      /* DEFINE THE MATH ROUTINES USED BY MODULES */
27     1      MQULD4: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
28     2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
29     2      END MQULD4;
30     1      MQUDV2: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
31     2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
32     2      END MQUDV2;

```

```

34 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
35 2      END MQUDV1;
36 1      MQUST1: PROCEDURE (IR$PTR,VALUE$PTR) EXTERNAL;
37 2      DECLARE (IR$PTR,VALUE$PTR) ADDRESS;
38 2      END MQUST1;

/* DEFINE THE MATH ACCUMULATOR STORAGE AREA */
39 1      DECLARE IR(18) BYTE EXTERNAL;

/* DEFINE THE COUNTER LOCATIONS */
40 1      DECLARE (LIQ$COUNT,BELT$COUNT) BYTE EXTERNAL;

$EJECT
/*****
* THIS PROGRAM IS USED TO GENERATE A FREQUENCY *
* OUTPUT USING THE ISBC 941 MODULE INSTALLED IN *
* SOCKET NUMBER 0. TO PROVIDE MAXIMUM RESOLU- *
* TION, FOUR PERIODS WILL BE USED. THE FREQUEN- *
* CY RANGES CORRESPONDING TO EACH PERIOD ARE: *
* RANGE          FREQ          RESOLUTION *
* 1          50 TO 165 HZ      2 HZ *
* 2          166 TO 225 HZ      3 HZ *
* 3          226 TO 285 HZ      3 HZ *
* 4          286 TO 550 HZ      6 HZ *
* THE SCALE FACTOR IS COMPUTED BY THE FORMULA: *
* SF=100000/((FREQ)*(RANGE FACTOR)) *
*****/

41 1      WEIGHBELT$MOTOR$DRIVE: PROCEDURE (FREQ) PUBLIC;

/* DECLARATION OF CONSTANT, 100,000 */
42 2      DECLARE HUNDRED$K(4) BYTE DATA (
          0A0H,086H,001H,000H );

/* DECLARATION OF ISBC941 PORT ENABLES */
43 2      DECLARE ENABLE$FREQ LITERALLY '01H';
44 2      DECLARE DISABLE$FREQ LITERALLY '00H';

/* DECLARATION OF ISBC 941 MEMORY LOCATION COMMANDS */
45 2      DECLARE WRRM$55 LITERALLY '055H';
46 2      DECLARE WRRM$74 LITERALLY '074H';

/* DECLARATION OF VARIABLES USED IN COMPUTATIONS */
47 2      DECLARE (RANGE,FREQA) BYTE;
48 2      DECLARE FREQ ADDRESS;

/* BEGIN COMPUTATION OF OUTPUT FOR FREQ > 48 HZ. */
49 2      IF FREQ > 49
          THEN DO;

/* ENABLE FREQUENCY OUTPUT */
51 3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
52 4      END;
53 3      OUTPUT(UPI$0$COMMAND) = SETOE;

```

```

54 3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
55 4      END;
56 3      OUTPUT(UPI$0$DATA) = ENABLE$FREQ;

      /* COMPUTATION OF FREQUENCY RANGE */
57 3      IF FREQ < 285
      THEN DO;
59 4      IF FREQ < 226
      THEN DO;
61 5      IF FREQ < 166
      THEN RANGE = 9;
63 5      ELSE RANGE = 5;
64 5      END;
65 4      ELSE RANGE = 3;
66 4      END;
67 3      ELSE RANGE = 2;

      /* LOAD MATH ACCUMULATOR WITH 100,000 */
68 3      CALL MQULD4 (.IR,.HUNDRED$K);

      /* TEST FOR MOTOR SHUTDOWN */
69 3      IF FREQ > 1
      THEN DO;

      /* DIVIDE BY FREQUENCY */
71 4      CALL MQUDV2 (.IR,.FREQ);

      /* DIVIDE BY RANGE FACTOR */
72 4      CALL MQUDV1 (.IR,.RANGE);

      /* GET TWO'S COMPLEMENT FOR ISBC 941 SCALE FACTOR */
73 4      CALL MQUST1 (.IR,.FREQA);
74 4      FREQA = NOT (FREQA + 1);
75 4      END;

      /* ADJUST FOR MOTOR STOP SIGNAL */
76 3      ELSE DO;
77 4      FREQA = 000H;
78 4      RANGE = 0FFH;
79 4      END;

      /* SEND NEW SCALE FACTOR TO DEVICE */
80 3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
81 4      END;
82 3      OUTPUT(UPI$0$COMMAND) = WRRM$55;

83 3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
84 4      END;
85 3      OUTPUT(UPI$0$DATA) = FREQA;

      /* SEND NEW PERIOD TO DEVICE */
86 3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
87 4      END;
88 3      OUTPUT(UPI$0$COMMAND) = WRRM$74;

```



```

89      3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
90      4      END;
91      3      OUTPUT(UPI$0$DATA) = RANGE;

/* END OF FREQUENCY OUTPUT MODE */
92      3      END;

/* HANDLE FREQUENCIES < 50 HZ. */
93      2      ELSE DO;

/* DISABLE FREQUENCY OUTPUT GENERATION */
94      3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
95      4      END;
96      3      OUTPUT(UPI$0$COMMAND) = SETOE;

97      3      DO WHILE ((INPUT(UPI$0$STATUS) AND IBF) <> 0);
98      4      END;
99      3      OUTPUT(UPI$0$DATA) = DISABLE$FREQ;

/* END OF ALTERNATE FREQUENCY OUTPUT */
100     3      END;

/* RETURN TO CALLING PROGRAM */
101     2      RETURN;

102     2      END WEIGHBELT$MOTOR$DRIVE;

$EJECT
/*****
* THIS PROGRAM GETS THE WEIGHBELT WEIGHT FROM THE
* NUMBER 1 ISBC 941 PROCESSOR. THE WEIGHT WILL BE
* RECEIVED AS A COUNT WHICH RANGES BETWEEN 0 AND
* 2000, CORRESPONDING TO A WEIGHT BETWEEN 0.0 AND
* 10.00 POUNDS. EACH COUNT RECEIVED HAS A VALUE
* OF 0.005 POUNDS.
*****/

103     1      WEIGHBELT$WEIGHT: PROCEDURE ADDRESS PUBLIC;

/* DECLARATIONS OF VARIABLES USED IN THE PROCEDURE */
104     2      DECLARE (LCOUNT,HCOUNT) BYTE;
105     2      DECLARE WEIGHT ADDRESS;

/* GET INPUT COUNT LOW BYTE */
106     2      DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
107     3      END;
108     2      OUTPUT(UPI$1$COMMAND) = RDR0;

109     2      DO WHILE ((INPUT(UPI$1$STATUS) AND OBF) = 0);
110     3      END;
111     2      LCOUNT = INPUT(UPI$1$DATA);

```

```

112 2 /* GET INPUT COUNT HIGH BYTE */
113 3 DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
114 2 END;
115 2 OUTPUT(UPI$1$COMMAND) = R0FC1;
116 3 DO WHILE ((INPUT(UPI$1$STATUS) AND OBF) = 0);
117 2 END;
118 2 HCOUNT = INPUT(UPI$1$DATA);
119 3 /* START NEXT WEIGHT SAMPLE PERIOD */
120 2 DO WHILE ((INPUT(UPI$1$STATUS) AND IBF) <> 0);
121 3 END;
122 2 OUTPUT(UPI$1$COMMAND) = LOOP;
123 2 /* CONVERT WEIGHT TO AN ADDRESS VALUE */
124 2 WEIGHT = HCOUNT;
125 2 WEIGHT = SHL(WEIGHT,8);
126 2 WEIGHT = WEIGHT + LCOUNT;
127 2 /* DIVIDE BY TWO TO CONVERT TO POUNDS */
128 2 WEIGHT = SHR(WEIGHT,1);
129 2 /* RETURN THE WEIGHTBELT WEIGHT */
130 2 RETURN WEIGHT;
131 2 END WEIGHBELT$WEIGHT;
132 2 $EJECT
133 2 /*****
134 2 * THIS PROCEDURE TRANSFERS THE WEIGHBELT SPEED TO *
135 2 * THE CALLING PROGRAM AND CLEARS THE COUNTER FOR *
136 2 * THE NEXT TEST. THE SPEED RESOLUTION PROVIDES *
137 2 * ONLY FIVE SPEED RANGES. *
138 2 *****/
139 1 WEIGHBELT$SPEED: PROCEDURE BYTE PUBLIC;
140 2 /* DECLARATIONS OF VARIABLES USED BY THE PROCEDURE */
141 2 DECLARE SPEED BYTE;
142 2 /* LATCH COUNTER BEFORE READING SPEED */
143 2 DISABLE;
144 2 /* GET COUNTER VALUE FROM COUNTER */
145 2 SPEED = BELT$COUNT;
146 2 /* CLEAR COUNTER FOR NEXT OPERATION */
147 2 BELT$COUNT = 0;
148 2 ENABLE;
149 2 /* RETURN DATA TO CALLING ROUTINE */
150 2 RETURN SPEED;
151 2 END WEIGHBELT$SPEED;

```

```

$EJECT
/*****
* THIS PROCEDURE PROVIDES COMMANDS TO THE STEPPER *
* MOTOR TO OPERATE THE CONTROL VALVE. IT WILL COM- *
* PUTE THE SIGNED MAGNITUDE REPRESENTATION FROM *
* THE TWO'S COMPLIMENT INPUT AND WILL ISSUE THE *
* APPROPRIATE STEP INCREMENT AND DIRECTION. A *
* FIXED STEP RATE OF 100 STEPS PER SECOND WILL BE *
* USED BY THE CONTROL DEVICE. *
*****/

135 1 LIQUID$VALVE$POSITION: PROCEDURE (POSITION) PUBLIC;

136 2 /* DECLARATIONS OF VARIABLES USED BY THE PROCEDURE */
      DECLARE POSITION BYTE;

137 2 /* DEFINITIONS OF TERMS USED IN COMPUTATIONS */
138 2 DECLARE STEP$RATE LITERALLY '005H';
      DECLARE REVERSE LITERALLY '080H';

139 2 /* IF NO MOVEMENT, SKIP OPERATIONS */
      IF POSITION <> 0
      THEN DO;

141 3 /* SUPPORT CONVERSION TO SIGNED MAGNITUDE NUMBER */
      IF POSITION > 127
      THEN DO;

143 4 /* GET MAGNITUDE OF MOVEMENT */
      POSITION = 256 - POSITION;

144 4 /* SET SIGN FOR CCW ROTATION */
145 4 POSITION = POSITION OR REVERSE;
      END;

146 3 /* VERIFY THAT QUEUE SPACE IS AVAILABLE */
147 4 DO WHILE ((INPUT(UPI$2$STATUS) AND QF) <> 0);
      END;

148 3 /* REQUEST DESIRED STEP RATE */
149 4 DO WHILE ((INPUT(UPI$2$STATUS) AND IBF) <> 0);
150 3 END;
      OUTPUT(UPI$2$DATA) = STEP$RATE;

151 3 /* REQUEST STEPPER MOVEMENT */
152 4 DO WHILE ((INPUT(UPI$2$STATUS) AND IBF) <> 0);
153 3 END;
154 3 OUTPUT(UPI$2$DATA) = POSITION;
      END;

155 2 /* RETURN TO CALLING PROGRAM */
      RETURN;

156 2 END LIQUID$VALVE$POSITION;

```

```

$EJECT
/*****
* THIS PROCEDURE TRANSFERS THE LIQUID FLOW RATE FROM *
* THE FLOW COUNTER TO THE CALLING PROGRAM. AFTER *
* READING, THE FLOW COUNTER WILL BE RESET TO FACILI- *
* TATE THE NEXT READING. THE LIQUID FLOW COUNT WILL *
* VARY BETWEEN 20 AND 240 PULSES IN EACH 200 MILLI- *
* SECOND SAMPLE INTERVAL. THIS WILL CORRESPOND TO *
* THE ACTUAL LIQUID FLOW RATE OF 10 TO 120 POUNDS *
* PER MINUTE. *
*****/

157 1 LIQUID$FLOW$RATE: PROCEDURE ADDRESS PUBLIC;

/* DECLARATION OF VARIABLES USED BY THE PROGRAM */
158 2 DECLARE TEMP BYTE;
159 2 DECLARE (FLOW,T$TWO,T$SXTN,T$THRTWO) ADDRESS;

/* LATCH COUNTER BEFORE READING FLOW */
160 2 DISABLE;

/* GET FLOW RATE VALUE FROM COUNTER */
161 2 TEMP = LIQ$COUNT;

/* CLEAR COUNTER FOR NEXT OPERATION */
162 2 LIQ$COUNT = 0;
163 2 ENABLE;

/* CONVERT TO POUNDS PER MINUTE */
164 2 FLOW = TEMP;
165 2 T$TWO = SHL(FLOW,1);
166 2 T$SXTN = SHL(T$TWO,3);
167 2 T$THRTWO = SHL(T$SXTN,1);
168 2 FLOW = T$TWO + T$SXTN + T$THRTWO;

/* RETURN FLOW RATE TO CALLING PROGRAM */
169 2 RETURN FLOW;

170 2 END LIQUID$FLOW$RATE;

$EJECT
/*****
* COUNTER FOR LIQUID FLOW RATE FROM LIQUID *
* FLOW METER. COUNT PULSE WILL GENERATE AN *
* INTERRUPT AT LEVEL 1. *
*****/

171 1 LIQ$CNT: PROCEDURE INTERRUPT 1 PUBLIC;

/* INCREMENT FLOW COUNT */
172 2 LIQ$COUNT = LIQ$COUNT + 1;

/* SEND END OF INTERRUPT */
173 2 OUTPUT (0ECH) = 020H;

```



```

*****
/* RETURN */
174 2 RETURN;
/* THIS PROCEDURE TRANSFERS LIQUID FLOW RATE FROM
* THE FLOW COUNTER TO THE CALLING PROGRAM AFTER
* READING THE FLOW COUNTER. THE LIQUID FLOW COUNTER WILL
* TAKE THE NEXT READING. VARY BETWEEN 0.2 AND 2.0 POUNDS
* PER MINUTE. CORRESPOND TO 1.0 TO 10.0 POUNDS
*****
* THIS PROCEDURE WILL PROVIDE AN EVENT COUN-
* TER TO HANDLE THE BELT MOTION DETECTOR.
* IT WILL OPERATE BY DIRECTING THE MOTION
* PULSE TO INTERRUPT 2.
*****/
175 2 END LIQ$CNT;
$EJECT
*****
176 1 BELT$CNT: PROCEDURE INTERRUPT 0 PUBLIC;
/* INCREMENT BELT MOVEMENT */
177 2 BELT$COUNT = BELT$COUNT + 1;
/* SEND END OF INTERRUPT */
178 2 OUTPUT (0ECH) = 020H;
/* RETURN */
179 2 RETURN;
/* CLEAR COUNTER FOR NEXT OPERATION */
180 2 END BELT$CNT;
181 1 END PROCESSOR$INTERFACE$MODULE;
/* CONVERT TO POUNDS PER MINUTE */
/* FLOW = TEMP;
T$TWO = SHL(FLOW,1);
T$XTN = SHL(T$TWO,1);
T$TWO = SHL(T$XTN,1);
FLOW = T$TWO + FLOW;
/* RETURN FLOW RATE TO CALLING PROGRAM
RETURN FLOW;
*****
MODULE INFORMATION:
CODE AREA SIZE = 0251H 593D
VARIABLE AREA SIZE = 0013H 19D
MAXIMUM STACK SIZE = 0008H 8D
400 LINES READ
0 PROGRAM ERROR(S)
END OF PL/M-80 COMPILATION
*****
/* COUNTER FOR LIQUID FLOW RATE FROM LIQUID
* FLOW METER. COUNT PULSE WILL GENERATE AN
* INTERRUPT AT LEVEL 1.
*****
LIQ$CNT: PROCEDURE INTERRUPT 1 PUBLIC;
/* INCREMENT FLOW COUNT */
LIQ$COUNT = LIQ$COUNT + 1;
/* SEND END OF INTERRUPT */
OUTPUT (0ECH) = 020H;

```

Designing and Assembling Microcomputer Systems Grows Easier

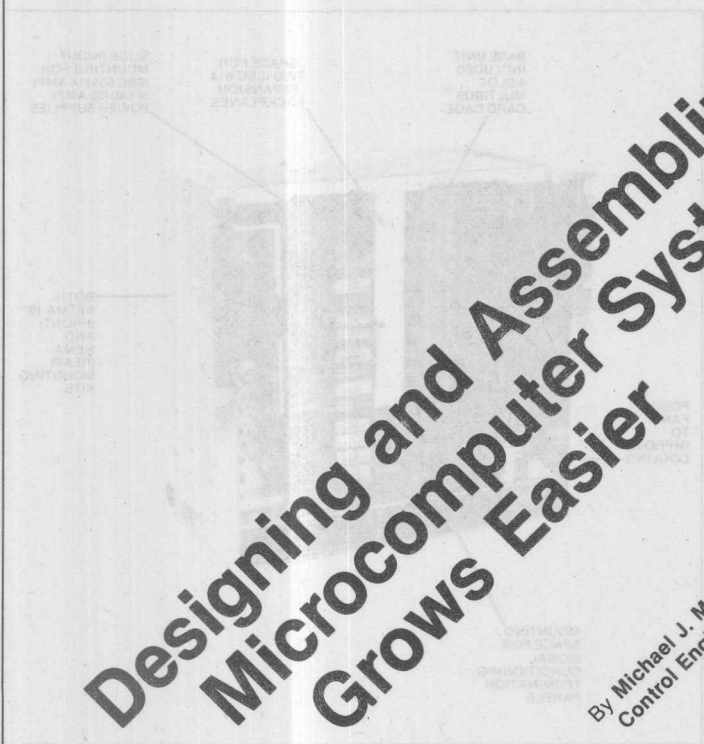
Although a single data bus standard yet under the microcomputer industry, Intel has introduced a single-board computer and supplementary boards that have greatly simplified the design of microcomputer systems. With Intel's new design, a single-board computer can be designed and assembled in a matter of days. The Intel 8080 microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days. The Intel 8080 microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days.

MICHAEL J. MCGOWAN, Control Engineering

But perhaps the most significant development in the microcomputer industry is the introduction of the Intel 8080 microcomputer. This microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days. The Intel 8080 microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days.

and not available in a general purpose board with extra features. But Intel's new design, a single-board computer can be designed and assembled in a matter of days. The Intel 8080 microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days.

Intel's new design, a single-board computer can be designed and assembled in a matter of days. The Intel 8080 microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days. The Intel 8080 microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days.



Designing and Assembling Microcomputer Systems Grows Easier

By Michael J. McGowan
Control Engineering/February 1979

Intel's new design, a single-board computer can be designed and assembled in a matter of days. The Intel 8080 microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days. The Intel 8080 microcomputer is the first of a new class of microcomputers that will make it possible to design and assemble microcomputer-based control systems in the matter of days.

Designing and Assembling Microcomputer Systems Grows Easier

Although a single data bus standard yet eludes the microcomputer industry, numerous manufacturers of single-board computer and supplementary boards have cast a hardware vote for the Multibus, Intel's microcomputer backplane which they originated in 1976. With a steady eye on the control industry market, Intel has designed a home to accommodate Multibus compatible equipment, the iCS-80 industrial chassis. It promises to significantly reduce the time and cost of assembling the housing and interface parts of a microcomputer-based control system. In this article, besides taking the first look at Intel's new chassis and signal conditioning panels, we've put together a comprehensive list of Multibus compatible equipment.

MICHAEL J. MCGOWAN, Control Engineering

After the development of single-board computers nearly three years ago, vendors moved quickly to seize a fraction of the market. It seemed at first that everything from memories to analog I/O boards had become available. With an astonishing suddenness, companies sprang up in Silicon Valley, Texas, New Jersey, and along the forested roadside of Rt. 128 outside of Boston. Late that year, we counted well over a hundred companies anxious to make their fortune selling the control engineer everything from one or two interface boards to complete microprocessor systems.

Then the pleasant dream became a nightmare. From power supply requirements to backplane pinouts, little was compatible. Even such an obvious thing as board size differed from vendor to vendor and many a hope for an ideal system was crushed in a pragmatic search for whatever would fit together.

Seven months ago in our June 1978 issue we noted that the number of microcomputer system manufacturers had dwindled to about 60 and since then, we find still fewer. Some, no doubt, were forced out for lack of reliability, though most, despite remarkably talented engineering, starved as the market saturated.

Large scale integration of microcomputer components has more than doubled the memory size of single-board computers. Sixteen bit word lengths will become commonplace in the next year as microcomputer performance begins to rival the mini-computer's and the lines of distinction between micros and minis fades.

The fight for a standard data bus drags on with leaders in the struggle but no winner. On the offensive, Pro-Log and Mostek jointly introduced the STD bus last autumn in an attempt to gain a greater market share by espousing decentralized system architectures. Their philosophy argues economics: the user should pay only for essential functions by selecting small, specialized boards

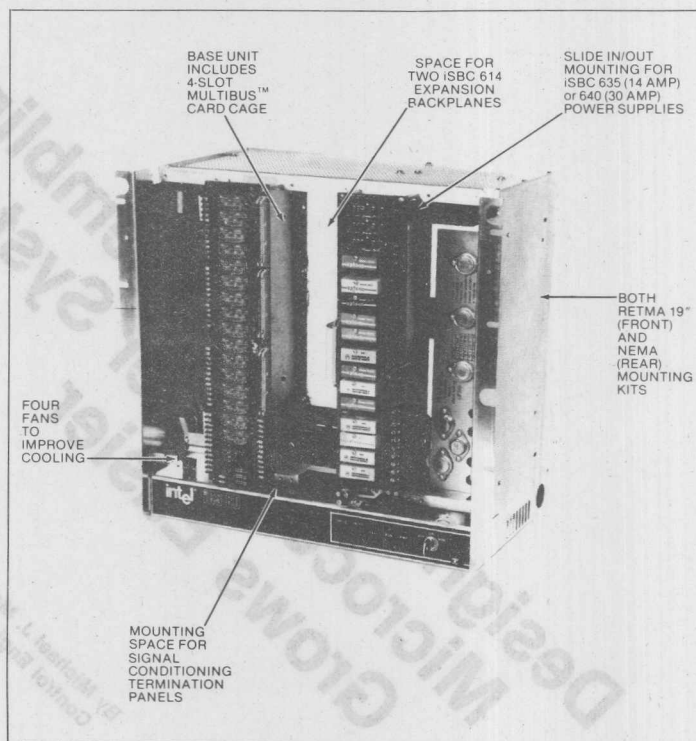
and not squander funds on a general purpose board with extra features.

Still, Intel, favoring more densely packed and versatile boards, continues to dominate the market while the Multibus retains its popularity. Today more 30 manufacturers produce over one hundred different boards based on that bus structure alone. Not that Intel enjoys the strict fidelity of its outside vendors; Digital Equipment Corporation, for instance, boasts some 17 companies providing boards to mate with the LSI-11 and LSI-112.

But perhaps alone among its competitors, Intel has recognized that the majority of its boards are being used in industrial applications and that the control system designer needs more than components.

An industrial chassis

A microcomputer system designer must choose components that are electro-mechanically compatible. To that end, Intel is introducing the iCS-80 industrial chassis and termination panels. It makes all Multibus-compatible CPU



MULTIBUS Compatible Boards and Vendors	Analog Input and Output Boards	Analog Signal Conditioning/ Screw Terminations	Communications Boards	Core Memory Boards	Digital Input/Output Boards	Digital Relay Boards	CPU Boards	DMA Controller Boards	Floppy Disk Controller Boards	Hard Disk Interface Boards	IEEE-488 Bus Interface Boards	Keyboard Controller Boards	Math Boards	Optical Isolator Boards (without screw terminations)	Optical Isolation With Screw Terminations	PROM Boards	RAM Boards	RAM/PROM or ROM	Synchro to Digital Boards	Tape (cartridge/cassette) Interface Boards	Tape (1/2") Interface Boards	Video Boards	Wire Wrap Boards
ADAC Corp.	•																						
Advanced Micro Computers							•						•										
Ampex				•																			
Analog Devices	•																						
Augat																							•
Burr-Brown	•				•																		
Computer Marketing	•								•	•							•				•	•	•
Data Translation	•																						
Datacube																	•	•	•				•
Datel Systems	•																						
Electronic Solutions																	•	•					
Garry Manufacturing																							•
HT Instruments																				•			
Hal Communications																							•
Heurikon	•						•		•									•	•				•
IDEAS	•								•		•		•						•				
Intel	•	•	•	•	•		•	•	•			•	•	•	•	•	•	•	•	•	•	•	•
Interphase			•						•	•													•
Matrox Electronic Systems																							•
Megalogic																	•	•	•		•		
Micro Memories																							
Micro Networks	•			•																			
MicroTec																	•						
Micro/Tel									•														
Monolithic Systems							•		•										•				•
Motorola																		•					
MUPRO																		•					
National Semiconductor	•		•	•	•		•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•
North Star Computers													•										
Pertec (ICOM)									•														
Relational Memory Systems							•																
Systems, Computers and Interfaces							•		•			•						•	•				•
Thomas Engineering Co.																							
Vector Electronic																							•
XEDAX												•										•	
ZIA Tech										•													

The advantage of the ICS-80 is that most of the interconnection and mechanical details for assembling a microcomputer-based control system have already been worked out.

The ICS-80 stands 15.75 inches high and can be mounted in a standard RETMA 19 inch rack, or in a NEMA cabinet secure from the industrial environment. The minimum layout consists of a four-slot Multibus card cage with provisions for adding two more cages to a maximum of twelve cards. The cages fit vertically, like records in a rack, to aid convection cooling and permit front access for insertion and maintenance.

On the right side of the chassis is room for either a 14 or 30 ampere power supply, the choice dictated by the application. The system will operate on either 115 or 230 volts with a range of 47 to 63 Hertz specified in anticipation of international service.

Cooling is assisted by four fans—three for the card cages and one for the power supply section. The intention here is to make the installation of additional fans unnecessary even after the system has expanded. The fans are expected to provide adequate cooling for most applications so supplementary air conditioning can be eliminated or at least minimized.

Signal conditioning

Three signal conditioning panels have been developed by Intel to simplify connections between the processing cards and the outside world. The principle is neatness, and with that follows reliability. Flat ribbon cables connect the signal conditioners to the processor cards, a safeguard from "which wire is which" and screwdriver slips in the vicinity of expensive boards. Field connections to the external inputs and outputs are made (presumably by electricians with big hands and reputations for being less than delicate) through rugged, screw-type barrier strips that accept wire as heavy as 14 AWG. The panels can mount either on RETMA cabinet brackets, NEMA wall spacers, or on the ICS-80 chassis itself.

Each signal conditioning card gives the user a variety of options. The ICS-910 analog signal conditioning/termination panel accepts up to 16 differential or 32 single ended input channels. The four 2-wire analog output channels might be connected to 4 to 20 mA current loops.

The digital signal conditioning termination panel, ICS-920, handles 24 two-wire input or output channels with signals up to 55 V, 300 mA. Inputs can be diode protected, and pads are provided for current limiters or voltage dividers. Optoisolators may be inserted in

ADAC Corp. Woburn, MA 617/935-6668	Advanced Micro Computers Santa Clara, CA 408/732-2400	Ampex El Segundo, CA 714/973-2970	Analog Devices Norwood, MA 617/329-4700	Augat Attleboro, MA 617/222-2202	Burr-Brown Tucson, AZ 602/655-8000	Computer Marketing Waltham, MA 617/894-7000	Data Translation Natick, MA 617/655-5300	Datacube Reading, MA 617/944-4600	Datel Systems Canton, MA 617/828-8000	Electronic Solutions San Diego, CA 714/292-0242	Garry Manufacturing New Brunswick, NJ 212/267-6844	HT Instruments Marina Del Rey, CA 312/822-4296	Hal Communications Urbana, IL 217/367-7373	Heurikon Madison, WI 608/255-9075	IDEAS Beltsville, MD 301/937-3600	Intel Aloha, OR 503/642-2563	Interphase Dallas, TX 214/238-0971
--	---	---	---	--	--	---	--	---	---	---	--	--	--	---	---	------------------------------------	--

Matrox Electronic Systems Montreal, Quebec 514/735-1182	Megalogic Brookville, OH 513/833-5222	Micro Memories Chatsworth, CA 213/998-0070	Micro Networks Worcester, MA 617/852-5400	MicroTec Sunnyvale, CA 408/733-2919	Micro/Tel St. Louis, MO 314/569-3450	Monolithic Systems Englewood, CO 303/770-7400	Motorola Austin, TX 512/928-6572	MUPRO Sunnyvale, CA 408/737-0500	National Semiconductor Santa Clara, CA 408/737-5262	North Star Computers Berkeley, CA 415/549-0858	Pertec (ICOM) Chatsworth, CA 213/998-1800	Relational Memory Systems San Jose, CA 408/248-6356	Systems, Computers and Interfaces Waltham, MA 617/899-2359	Thomas Engineering Co. Concord, CA 415/686-3041	Vector Electronic Sylmar, CA 213/365-9661	XEDAX Alameda, CA 415/521-6600	ZIA Tech Cupertino, CA 408/996-7082
---	---	--	---	---	--	---	--	--	---	--	---	---	--	---	---	--------------------------------------	---

the DIP sockets for high voltage isolation or jumpers may be used instead when the input is TTL. Similarly, output sockets accept jumpers for direct TTL output, DIP optoisolators for transient suppression, or integrated circuit (open collector) drivers for high voltage to high current outputs. Activity on each channel is indicated by LEDs.

The ac signal conditioning (solidas) termination panel, ICS-930, will actually work with ac or dc on its 16 channels. The user supplies optoisolators for input isolation and optically-isolated solid

state relays for output isolation. Mounting pads for customer-supplied MOVs or snubber networks are included. As before, a fuse gives overload protection and LEDs indicate channel activity.

The advantage of all this is that by plugging in some components and perhaps inserting a few resistors and capacitors, the interface units can be tailored to a particular application. Since many mechanical and electrical connection problems have already been solved, a customized unit can be built with minimum effort. □

USER'S PROGRAM LIBRARY

- Programs for 8048, 8080/8085, and 8086/8087/8088 Processors
- Accepted Program Submittals Entitle You to a Free Membership or Free Program Package
- Worldwide Offices to Serve You
- Diskettes, Paper Tapes, and Listings Available for Library Programs
- Program Library Catalog Offering Hundreds of Programs
- Updates of New Programs Sent During Subscription Period

Insite, Intel's Software Index and Technology Exchange Library, is a varied collection of programs and routines that have been written by users of Intel microcomputers, single-board computers, and development systems. This expanding library of programs covers a broad range of software tools that includes monitors, conversion routines, peripheral drivers, translators, math packages, and even games. As a library member, you can acquire a copy of any program within the library on any of its available types of media. By taking advantage of the availability of existing library programs, numerous hours of coding and debugging time can be saved and routine or redundant programming operations can be eliminated. The Insite Program Library also serves as a learning tool for individuals unfamiliar with assembly or high-level languages associated with Intel's family of microcomputers.

Membership. Membership in Insite is available on an annual basis. Intel customers may become members through an accepted program contribution or paid membership fee.

Program Submittals. The Insite Library is built on program submittals contributed by users. Customers are encouraged to submit their programs. For each accepted program, submitters will receive a choice of three free programs, or free membership with Insite for one year. (Forms and submittal requirements are attached.)

Program Library Service. PAPER TAPES, DISKETTES OR SOURCE LISTINGS are available for every program in Insite. Diskettes are available on single or double density. Membership is required to purchase programs.

Insite™ Program Library Catalog. Each member will be sent the Program Library Catalog consisting of an abstract for each program indicating the function of the routine, required hardware and software, and memory requirements.

Insite members will be updated with abstracts of new programs submitted to the Library during the subscription period. For catalog and yearly subscription fee please refer to the Intel OEM Price List or contact the nearest Insite or Intel Sales Office.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products: BXP, CREDIT, I, ICE, ICS, Im, Insite, Intel, Intelvision, Intellex, iMMX, iDSP, IPDS, iRMX, ISBC, ISBX, Library Manager, MCS, MULTIMODULE, Megachassis, Micromainframe, Micromap, MULTIBUS, Multichannel, Plug-A-Bubble, PROMPT, Promware, RMX/80, System 2000, UPI, and the combination of ICS, iRMX, ISBC, ISBX, ICE, MCS, or UPI and a numerical suffix. Intel Corporation Assumes No Responsibility for the use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Patent Licenses are implied. ©INTEL CORPORATION, 1982. ORDER NUMBER: 121707-001



INSITE™ USER'S PROGRAM LIBRARY

SUBMITTAL REQUIREMENTS

Programs submitted for Insite review must follow the guidelines listed below:

Programs must be written in a language capable of compilation and assembly by the currently-supported version of an Intel standard compiler/assembler. Accepted languages are documented in the following manuals available through Intel's Literature Department.

- BASIC-80 Reference Manual, Order No. 400710
- iCIS-COBOL Language Reference Manual, Order No. 409115
- FORTRAN-80 Programming Manual, Order No. 400625
- FORTRAN-86 User's Guide, Order No. 400645
- Pascal-80 User's Guide, Order No. 400660
- Pascal-86 User's Guide, Order No. 400680
- PL/M-80 Programming Manual, Order No. 401700
- PL/M-86 Programming Manual, Order No. 402300
- MCS-48 and UPI-41A Assembly Language Manual, Order No. 305000
- MCS-86 Macro Assembly Language Reference Manual, Order No. 402105
- 8080/8085 Assembly Language Programming Manual, Order No. 401100
- 8086/8087/8088 Macro Assembly Language Reference Manual for 80/85 Based Development System, Order No. 402165
- 8086/8087/8088 Macro Assembly Language Reference Manual for 80/86 Based Development System, Order No. 402157
- 8089 Assembly Language Reference Manual, Order No. 305000
- Microsoft* BASIC Compiler Reference Manual, Order No. 121805
- Microsoft* BASIC-80 Reference Manual, Order No. 121806
- Microsoft* BASIC Reference Book, Order No. 121857
- Microsoft* FORTRAN-80 Reference Manual, Order No. 121798
- Microsoft* FORTRAN-80 User's Manual, Order No. 121799
- Microsoft* M/Sort Reference Manual, Order No. 121809
- Microsoft* Utility Software Manual, Order No. 121797

A well-documented source code furnished on an ISIS-formatted diskette, CP/M*-formatted diskette, or ASCII-coded paper tape.

A source listing of the program must be included. This must be the output listing of a compilation or an assembly. No consideration will be given to incomplete programs or duplications of programs already in the Library.

A link and locate listing.

A demonstration program which assures the validity of the contributed program must be included. This must show the accurate operation of the program.

A complete submittal form.

Licensed software or copyrighted material must be accompanied by a written release from the appropriate, authorized person.

*Microsoft is a trademark of Microsoft, Inc.

*CP/M is a registered trademark of Digital Research, Inc.



INSITE™ USER'S PROGRAM LIBRARY SUBMITTAL FORM

Processor

☐ 8048 ☐ 8080/8085 ☐ 8086/8087/8088 ☐ Other _____

Indicate the Microcomputer Development System series model the program was created on by checking the appropriate box, and identify other Microcomputer Development System series models the program may be compatible with.

Program
Title

Function

Required
Hardware

Required
Software

Input
Parameters

Output
Results

Registers Modified:

RAM Required:

ROM Required:

Maximum Subroutine Nesting Level:

Assembler/Compiler Used:

Programming Language:

Programmer:

Company:

Address:

City:

State:

Telephone:

ACKNOWLEDGEMENT AND AGREEMENT

To the best of my knowledge, I have the right to contribute this program material without breaching any obligation concerning nondisclosure of proprietary or confidential information of other persons or organizations. I am contributing this program material on a nonconfidential nonobligatory basis to the Insite User's Library for inclusion in its program library, and I agree that the Library may use, duplicate, modify, publish, and sell the program material without obligation or liability of any kind. The Insite User's Library may publish my name and address, as the contributor, to facilitate user inquiries pertaining to this program material.

Signature _____

Date _____

INSITE™ USER'S PROGRAM LIBRARY

MEMBERSHIP FORM

I WISH TO BECOME A MEMBER OF INSITE. ENCLOSED IS:

- ☐ CHECK/MONEY ORDER
☐ PURCHASE ORDER
☐ PROGRAM SUBMITTAL

MEMBER NAME: _____

COMPANY: _____

ADDRESS: _____

TELEPHONE: _____

REFER TO THE INSITE PRICE LIST FOR ANNUAL MEMBERSHIP FEE.

RETURN COMPLETED FORM TO THE NEAREST INSITE OFFICE:

NORTH AMERICA

Intel Corporation
 3065 Bowers Avenue
 Santa Clara, California 95051
 ATTN: Insite User's Program Library
 Telephone: 408-987-8080

THE ORIENT

Intel Japan K.K.
 5-6 Tohkohdai, Toyosato-cho,
 Tsukuba-gun, Ibaraki, 300-26, Japan
 ATTN: Insite User's Program Library
 Telephone: 029747-8511

EUROPE

Intel Corporation S.A.R.L.

5 Place de la Balance

Silic 223

94528 Rungis Cedex, France

ATTN: Insite User's Program Library

Telephone: 0687-22-21

Intel Semiconductor GmbH

Seidlstrasse 27

8000 Muenchen 2

West Germany

ATTN: Insite User's Program Library

Telephone: 089-5389-1

Intel Corporation (U.K.) Ltd.

Pipers Way

Swindon SN3 LRJ

Wiltshire, England

ATTN: Insite User's Program Library

Telephone: 0793-488-388



DIGITAL RESEARCH INC. CP/M* 2.2 OPERATING SYSTEM

- **High-performance, single-console operating system**
- **Simple, reliable file system matched to microcomputer resources**
- **Table-driven architecture allows field reconfiguration to match a wide variety of disk capacities and needs**
- **Extensive documentation covers all facts of CP/M applications**
- **More than 1,000 commercially available compatible software products**
- **General-purpose subroutines and table-driven data-access algorithms provide a truly universal data management system**
- **Upward compatibility from all previous versions**

CP/M 2.2 is a monitor control program for microcomputer system and application uses on Intel 8080/8085-based microcomputer (see the *CP/M-86* Operating System* data sheet for information on CP/M for Intel 8086/8088-based systems). CP/M provides a general environment for program execution, construction, storage, and editing, along with the program assembly and check-out facilities.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this system, a large number of distinct programs can be stored in both source- and machine-executable form.

CP/M also supports a powerful context editor, Intel-compatible assembler, and debugger subsystems. Nearly all personal software programs can be bought configured to run under CP/M, several of which are available from Intel.

FEATURES

CP/M is logically divided into four distinct modules:

BIOS—Basic I/O System

- Provides primitive operations for access to disk drives and interface to standard peripherals (teletype, CRT, paper tape reader/punch, bubble memory, and user-defined peripherals)
- Allows user modification for tailoring to a particular hardware environment

BDOS—Basic Disk Operating System

- Provides disk management for one to sixteen disk drives containing independent file directories
- Implements disk allocation strategies for fully dynamic file construction and minimization of head movement across the disk

- Uses less than 4K of memory allowing plenty of memory space for applications programs
- Uses less than 4K of memory
- Makes programs transportable from system to system
- Entry points include the following primitive operations which can be programmatically accessed:
 - SEARCH Look for a particular disk file by name
 - OPEN Open a file for further operations
 - CLOSE Close a file after processing
 - RENAME Change the name of a particular file
 - READ Read a record from a particular file
 - WRITE Write a record to a particular file
 - SELECT Select a particular disk drive for further operations

CCP—Console Command Processor

- Provides primary user interface by reading and interpreting commands entered through the console
- Loads and transfers control to transient programs, such as assemblers, editors, and debuggers
- Processes built-in standard commands including:

ERA	Erase specified files
DIR	List file names in the directory
REN	Rename the specified file
SAVE	Save memory contents in a file
TYPE	Display the contents of a file on the console

TPA—Transient Program Area

- Holds programs which are loaded from the disk under command of the CCP
- Programs created under CP/M can be checked out by loading and executing these programs in the TPA
- User programs, loaded into the TPA, may use the CCP area for the program's data area
- Transient commands are specified in the same manner as built-in commands
- Additional commands can be easily defined by the user
- Defined transient commands include:

PIP	Peripheral Interchange Program —implements the basic media transfer operations necessary to load, print, punch, copy, and combine disk files; PIP also performs various reformatting and concatenation functions. Formatting options include parity-bit removal, case conversion, Intel hex file validation, subfile extraction, tab expansion, line number generation, and pagination
ED	Text Editor—allows creation and modification of ASCII files using extensive context editing commands: string substitution, string search, insert, delete and block move; ED allows text to be located by context, line number, or relative position with a macro command for making extensive text changes with a single command line

ASM

Fast 8080 Assembler—uses standard Intel mnemonics and pseudo operations with free-format input, and conditional assembly features

DDT

Dynamic Debugging Tool—contains an integral assembler/disassembler module that lets the user patch and display memory in either assembler mnemonic or hexadecimal form and trace program execution with full register and status display; instructions can be executed between breakpoints in real-time, or run fully monitored, one instruction at a time

SUBMIT

Allows a group of CP/M commands to be batched together and submitted to the operating system by a single command

STAT

Lists the number of bytes of storage remaining on the currently logged disks, provides statistical information about particular files, and displays or alters device assignments

LOAD

Converts Intel hex format to absolute binary, ready for direct load and execution in the CP/M environment

SYSGEN

Creates new CP/M system disks for back-up purposes

MOVCPM

Provides regeneration of CP/M systems for various memory configurations and works in conjunction with SYSGEN to provide additional copies of CP/M

BENEFITS

- Easy implementation on any computer configuration which uses an Intel 8080/8085 Central Processing Unit (see the CP/M-86 data sheet for CP/M applications on the iAPX86 CPU)
- iPDS version supports bubble memory option as an additional diskette drive. Also allows diskette duplication with a single drive
- Extensive selection of CP/M-compatible programs allows production and support of a comprehensive software package at low cost
- Field programmability for special-purpose operating system requirements
- Upward compatibility from previous versions of CP/M release 1

- Provides field specification of one to sixteen logical drives, each containing up to eight megabytes
- Files may contain up to 65,536 records of 128 bytes each but may not exceed the size of any single disk
- Each disk is designed for 64 distinct files—more directory entries may be allocated if necessary
- Individual users are physically separated by user numbers, with facilities for file copy operations from one user area to another
- Relative-record random-access functions provide direct access to any of the 65,536 records of an eight-megabyte file

SPECIFICATIONS

Hardware Required

- Model 800 with 720 kit
- DS 235 kit or MDS 225 with 720 kit (integral drive supported except as system boot device)
- iPDS Personal Development System
- Optional:
 - RAM up to 64K
 - Additional floppy disk drives
 - Single density via 201 controller
 - Bubble memory and optional Shugart 460 5¼" disk drive for iPDS

Documentation Package

Title

- CP/M 2.2 documentation consisting of 7 manuals:
- An Introduction to CP/M Features and Facilities
 - CP/M 2.2 User's Guide
 - CP/M Assembler (ASM) User's Guide
 - CP/M Dynamic Debugging Tool (DDT) User's Guide
 - ED: A Context Editor for the CP/M
 - Disk System User's Manual
 - CP/M 2 Interface Guide
 - CP/M 2 Alteration Guide

Shipping Media

(Specify by Alpha Character when ordering.)

- A—single density (IBM 3740/1 compatible)
- B—double density
- F—double-sided, double density 5¼" floppy (iPDS format)

Order Code

See Price List

Product Description

CP/M (Control Program for Microcomputers) is a disk-based operating system for the Intel 8080/8085-based systems. CP/M provides a general environment for program development, test, execution and storage. CP/M storage is available via a comprehensive, named-file structure supporting both sequential and random access. CP/M support tools include a Text Editor, a debugger, and an 8080/8085 assembler.

SUPPORT:

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required.

*CP/M is a registered trademark of Digital Research, Inc.

*CP/M-86, MP/M, CP/NET and MP/NET are trademarks of Digital Research, Inc.



DIGITAL RESEARCH INC. CP/M-86* OPERATING SYSTEM

- **High-performance, single-user operating system for 16-bit computers based on Intel's iAPX-86 (8086) and compatible iAPX-88 (8088) microprocessors**
- **CP/M-86 files are completely compatible with versions of CP/M* used with Intel 8-bit 8080- and 8085-based microcomputer systems**
- **Supports up to 16 logical drives, each containing up to eight megabytes for a total of 128 megabytes of on-line storage**
- **CP/M-86 manages up to 1 Mbyte of on-line RAM storage for full support of programs as large and as powerful as those found on minicomputers**
- **The standard CP/M-86 package includes an assembler, an interactive debugger, and additional utilities for complete program development**

CP/M-86 is a single-user operating system designed especially for 16-bit computers based on the Intel iAPX-86 (8086) and compatible iAPX-88 (8088) microprocessors. The system fully utilizes the one megabyte (1,048,576 bytes) of main memory available for application programs.

CP/M-86 occupies only 11K of memory with a floppy disk-based I/O System occupying about 1K, depending on the number of peripherals supported. The remainder of the 8086/8088 address space may be defined by the user. Flexibility is provided by the system's ability to reside anywhere in memory, and it can be relocated by the user without changing the operating system.

Because CP/M-86 files are completely compatible with CP/M for Intel 8-bit microprocessors, there is an easy upgrade of existing CP/M applications software to 16-bit iAPX-86 (8086)-based systems.

FEATURES

Major features of the CP/M-86 operating system include:

Modular Design

Based on a proven, modular design, the system includes the:

- CCP: Console Command Processor
The CCP is the human interface to the operating system and performs decoding and execution of user commands
- BDOS: Basic Disk Operating System
The BDOS is the logical, invariant portion of the operating system; it supports a named file system with a maximum of 16 logical drives, containing up to 8 megabytes each for a potential of 128 megabytes of on-line storage

—BIOS: Basic Input/Output System

The physical variant portion of the operating system, BIOS contains the system-dependent input/output device handlers

CP/M Compatibility

CP/M-86 files are completely compatible with CP/M for 8080- and 8085-based microcomputer systems. This simplifies the conversions of software developed under CP/M and allows full advantage of 16-bit 8086-based systems.

The user will notice no significant difference between CP/M and CP/M-86. Commands such as DIR, TYPE, REN, ERA, PIP, ED, and STAT respond the same way in both systems.

CP/M-86 uses the 8086 registers corresponding to 8080 registers for system call and return parameters to further simplify software transport. The execution environment in CP/M-86 allows code and data segments to overlap, making the mixture of code and data that often appears in 8-bit applications acceptable to the 8086.

File Management

CP/M-86 can support up to 16 logical drives, each containing up to eight megabytes, for a maximum of 128 megabytes of on-line storage. Any one file can reach the full drive size, with space dynamically allocated and released. Each device has a directory of file control blocks that map the physical location of each file on the disk. Disk definition tables in the BIOS translate this logical drive, directory, and file structure to the physical characteristics of the disk. This file system is identical to the file system of CP/M.

Memory Management

CP/M-86 can reside anywhere in memory and is easily located to minimize dependence on absolute addresses. Multiple programs may reside in memory simultaneously. Also, a transient program may load additional programs for execution under its own control. Up to a total of 8 programs may use non-contiguous memory areas which are managed through a user-defined memory configuration table.

Basic I/O System (BIOS) Organization

The distribution version of CP/M-86 is set up for operation with the Intel iSBC 86/12A and an Intel 204 diskette controller. All hardware dependencies are, however, concentrated in subroutines which are collectively referred to as the Basic I/O System, or BIOS. A CP/M-86 system implementor can modify these subroutines to tailor CP/M-86 to fit nearly any 8086 or 8088 operating environment.

To simplify the preparation of a custom BIOS, a source listing of a working BIOS, a skeleton for a custom module, and cross-development utilities are supplied. The cross-development programs allow development of custom BIOS and CP/M-86 applications software on an 8-bit CP/M system.

PROM Loader

For users who have the iSBC 86/12 hardware configuration, there is an optional PROM Loader. The

firmware brings the CP/M-86 loader into the system and sets up the hardware to initialize CP/M-86.

Utilities

CP/M-86 is supplied with eight utilities:

PIP

The Peripheral Interchange Program provides file transfer between devices and disk files and performs various reformatting and concatenation functions. Formatting options include parity-bit removal, case conversion, Intel "hex" file validation, subfile extraction, tab expansion, line number generation, and pagination.

ED

The CP/M-86 Text Editor allows creation and modification of ASCII files using extensive commands: string substitution, string search, insert and delete. ED allows text to be located by context, line number, or relative position with a macro command for making extensive text changes with a single command line.

ASM-86

ASM-86, the CP/M-86 Assembler is an 8086 assembler using standard Intel mnemonics. It also allows users to define unique instructions with the code-macro facility. ASM-86 is supplied in two forms: an 8086 cross assembler designed to run under CP/M (an 8-bit system), and an 8086 assembler designed to run under CP/M-86.

DDT-86

The CP/M-86 Dynamic Debugging Tool allows the user to test and debug programs interactively in a CP/M-86 environment. The command set allows users to trace program execution with full register and status display. DDT-86 contains an integral assembler/disassembler module that lets users patch and display memory in assembler mnemonic form.

SUBMIT

Allows a group of CP/M-86 commands to be batched together and submitted to the operating system by a single command.

STAT

Lists the number of bytes of storage remaining on the currently logged disks, provides statistical information about particular files, and displays or alters device assignments.

GENCMD and LMCMD

GENCMD and LMCMD are used to produce CMD



memory image files suitable for execution under CP/M-86. The GENCMD utility processes Intel files, which may be produced either by Digital Research's ASM-86 or by Intel's OH86 utility. LMCMD processes Intel L-module files resulting from the standard Intel LOC86 Object Code Relocator Utility. These commands allow generation of new utilities.

BENEFITS

- Designed especially for full advantage of Intel iAPX-86/12A (8086)-based computer systems
- Complete compatibility with versions of CP/M for Intel 8080/8085-based microcomputer systems makes it easy to upgrade existing CP/M applications software to preserve software investment (see the CP/M Operating System data sheet for additional CP/M benefits)
- Cross-development tools, including the ASM-86 assembler and the GENCMD utility, can reside on a 8080/8085-based CP/M system; with these tools, the programmer can assemble a custom BIOS program and generate a loadable object file that runs on the target system
- Proven modular design occupies a small amount of memory to give maximum user-defined space

SPECIFICATIONS

Hardware Required

- iSBC 86/12A
- iSBC 204
- CRT

Optional:

- up to 1 megabyte RAM
- up to 16 disk drives (8 megabytes each)

Documentation Package

Description

CP/M-86 system, program and user's guides

SUPPORT:

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required.

*CP/M is a registered trademark of Digital Research, Inc.

CP/M-86 is a trademark of Digital Research, Inc.

for application programs in the available one-megabyte main memory.

- Allows up to 128 megabytes of on-line magnetic storage
- CP/M-86 manages up to 1 Mbyte of on-line RAM storage for full support of programs as large and as powerful as those found on minicomputers
- Allows multiple programs in memory for transient program execution; multiple programs may use non-contiguous memory areas for application programs
- ASM-86 and DDT-86 provide the basic tools for assembly language development and program debugging using the iAPX-86/12A (8086) microprocessor
- CP/M-86 is a complete system including the operating system and development tools; a comprehensive set of manuals describes system operation, customization, interfacing of applications programs, listings of sample programs, and operation of the assembler, debugger, and editor
- The CP/M-86 package includes full documentation for the product; documentation is also available separately; a CP/M-86 PROM loader is available for the Intel Single-Board Computer

Shipping Media

(Specify by Alpha Character when ordering.)

- A— Single Density (Intellec compatible, IBM 3740/1 format)

Order Code

Description

- | | |
|------------|---|
| SD12CPM86A | CP/M-86 is a single-user operating system for the Intel 8086 and 8088 microprocessors. CP/M-86 is modular in design, occupies a small amount of memory, and is completely compatible with earlier CP/M 2.2 for 8-bit microprocessors. As a part of the CP/M-86 package, Digital Research provides an assembler, interactive debugging tool, and other development utilities to allow users to copy files, edit files and generate memory image files. |
| CPM-86 1.0 | |



MICROSOFT INC.
MACRO-80 UTILITY



MICROSOFT*, INC. MACRO-80 UTILITY SOFTWARE PACKAGE

- Includes the MACRO-80 macro assembler, LINK-80 linking loader, and CREF-80 cross-reference facility
- Supports a complete, Intel-standard MACRO facility, including IRP, IRPC, REPEAT, local variables, and EXITM
- Supports conditional assembly, including testing of assembly pass, symbol definition, and parameters to MACROS
- Code is assembled in relocatable modules for easy manipulation by the LINK-80 linking loader

- Assembly rate of over 1000 lines per minute
- Provides "big computer" assembler features without sacrificing speed or memory space
- Provides a complete set of listing controls
- LINK-80 loads relocatable modules at user-specified locations
- CREF-80 cross-reference facility alphabetizes program variables and shows where each is defined and referenced

The Microsoft Utility Software Package is a complete system for developing assembly language programs, routines, and subroutines. The Utility Software Package includes the MACRO-80 macro assembler, the LINK-80 linking loader, and the CREF-80 cross-reference facility. The CP/M* version also includes the LIB-80 Library Manager.

The Utility Software Package is supplied with all Microsoft compilers to provide assembly language subroutine support to main programs in the high-level programming languages. The LINK-80 linking loader is used by all Microsoft compilers for linking and loading compiled relocatable modules. Thus, LINK-80 allows the programmer to link together relocatable modules from different Microsoft languages.

FEATURES

MACRO-80 Macro Assembler

MACRO-80 incorporates almost all "big computer" assembler features without sacrificing speed or memory space. The assembler supports a complete, Intel-standard macro facility, including IRP, IRPC, REPEAT, local variables, and EXITM. Macro names take precedence over instruction mnemonics and pseudo operations. Nesting of macros is limited only by memory. Code is assembled in relocatable modules that are easily manipulated with the flexible linking loader. Conditional assembly capability is greatly enhanced by an expanded set of conditional pseudo operations that include testing of assembly pass, symbol definition, and parameters to macros. Conditionals may be nested up to 255 levels.

More MACRO-80 features:

- Comment blocks
- Variable input radix from base 2 to base 16
- Octal or hex listings
- INCLUDE statement assembles an alternate source file into the current program
- PRINTX statement for printing assembly or diagnostic messages
- PHASE/DEPHASE statements allow code to reside in one area of memory but execute in another
- Complete set of listing controls

LINK-80 Linking Loader

With LINK-80, any number of programs may be loaded with one command, relocatable modules may be loaded in user-specified locations, and external references between modules are resolved automatically by the loader. The loader also performs library searches for system subroutines and generates a memory load map showing the locations of the main program and subroutines.

SPECIFICATIONS

Operating Environment

MACRO-80 resides in approximately 19K bytes of memory. LINK-80 resides in approximately 14K bytes of memory. CREF-80 requires about 6K bytes. The MACRO-80 Utility Software Package is compatible with the CP/M* operating system.

Required Hardware

Intellec® Microcomputer Development System

- Model 800 or
- Series II
- iPDS (Personal Development System)
- minimum of 1 diskette drive

Required Software

CP/M Operating System or MP/M-II* Operating System.

CREF-80 Cross-Reference Facility

The Cross-Reference Facility that is included with the Utility Software Package supplies a convenient alphabetic list of all program variable names, along with line numbers where they are referenced and defined.

Documentation Package

One copy of each manual is supplied with the software package.

Description

Microsoft Utility Software Manual

Shipping Media

(Specify by Alpha Character when ordering.)

- A—Single Density (Intellec® compatible, IBM 3740/1 format)
- B—Double Density (Intellec® compatible, M²FM format)
- F—Double-sided, Double Density 5¼" Floppy (iPDS format)

ORDERING INFORMATION

Order Code	Description
SD106CPM80A	Microsoft MACRO-80 Utility Software Package, CP/M version (Single-Density Diskette)
SD106CPM80B	Microsoft MACRO-80 Utility Software Package, CP/M version (Double-Density Diskette)
SD106CPM80F	Microsoft MACRO-80 Utility Software Package, CP/M version (iPDS Format)

SUPPORT:

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required.

*Microsoft is a trademark of Microsoft, Inc.

*CP/M is a registered trademark of Digital Research, Inc.

*MP/M-II is a trademark of Digital Research, Inc.



MICROSOFT INC.
BASIC-80 INTERPRETER



MICROSOFT*, INC. BASIC-80 INTERPRETER SOFTWARE PACKAGE

- Compatible with other Microsoft BASIC compilers and interpreters
- Sophisticated string handling and structured programming features for applications development
- Direct transfer of BASIC programs to the 8085, 8086 and 8088
- Random and sequential file manipulation where random file record length is user-definable
- Read or write memory location capabilities
- Meets the requirements for the ANSI subset standard for BASIC, and supports many enhancements
- Extensive text editing features built-in
- Automatic line number generation and renumbering
- Supports assembly language subroutine calls
- Trace facilities for easier debugging

BASIC Release 5.0 from Microsoft is an extensive implementation of BASIC. Microsoft BASIC gives users what they want from a BASIC—ease of use plus the features that are comparable to a minicomputer or large mainframe.

BASIC-80 meets the requirements for the ANSI subset standard for BASIC, as set forth in document BSRX3.60-1978. It supports many unique features rarely found in other BASICs.

FEATURES

- Four variable types: Integer (−32768, +32767), String (up to 255 characters), Single-Precision Floating Point (7 digits), Double-Precision Floating Point (16 digits).
- Formatted output using the PRINT USING facility, including asterisk fill, floating dollar sign, scientific notation, trailing sign, and comma insertion.
- Trace facilities (TRON/TROFF) for easier debugging.
- Direct access to I/O ports with the INP and OUT functions.
- Error trapping using the ON ERROR GOTO statement.
- Extensive program editing facilities via EDIT command and EDIT mode subcommands.
- PEEK and POKE statements to read or write any memory location.
- Assembly language subroutine calls (up to 10 per program) are supported.
- Automatic line number generation and renumbering, including reference line numbers.
- IF/THEN/ELSE and nested IF/THEN/ELSE constructs.
- Matrices with up to 255 dimensions.
- Supports variable-length random and sequential disk files with a complete set of file manipulation statements: OPEN, CLOSE, GET, PUT, KILL, NAME, MERGE.
- Boolean operators OR, AND, NOT, XOR, EQV, IMP.



BASIC-80 Commands, Statements, Functions

AUTO
LIST
NULL
TROFF
CLEAR
LOAD

RENUM
WIDTH
CONT
MERGE
RUN
DELETE

NAME
SAVE
EDIT
NEW
TRON

Program Statements

CALL
GOSUB
END
GOTO
STOP
WHILE/
WEND
CHAIN
DEF USR
LET
REM

RANDOMIZE
COMMON
DEF FN
ERROR
POKE
RESUME
SWAP
DEFDBL
DEFSTR
DEF SNG
DEFINT

RETURN
WAIT
ON GOSUB
DIM
FOR/NEXT/
STEP
IF/THEN/
ELSE
ON ERROR
GOTO
OPTION BASE

Input/Output Statements and Functions

CLOSE
KILL
OUT
RESTORE
READ
TAB
DATA
LINE
INPUT
PRINT
WRITE
LPRINT

GET
POS
FIELD
LSET/RSET
PRINT
USING
LOC
MKI\$
MK\$
MKD\$
LLIST
LPOS

NAME
PUT
EOF
SPC
INKEY\$
INPUT
OPEN
CVD
CVI
CVS

Arithmetic Functions

ABS
INT
SGN
ATN
EXP

SIN
CDBL
CSNG
CINT
SQR

LOG
FIX
COS
RND
TAN

String Functions

ASC
LEN
STRING\$
CHR\$
LEFT\$

STR\$
HEX\$
OCT\$
VAL

INSTR
RIGHT\$
MID\$
SPACE\$

Operators

=
^
<
>
-
/

*
=<
+
< >
\
>=

XOR
NOT
EQV
MOD
IMP
OR
AND

Special Functions

ERL
USR

ERR
FRE

VARPTR
PEEK

SPECIFICATIONS

Operating Environment

The standard disk version of Microsoft BASIC-80 occupies 24K bytes of memory. Microsoft BASIC-80 Interpreter is compatible with Intel's ISIS operating system or CP/M* operating system.

Required Hardware

Intellec Microcomputer Development System

- Model 800 or
- Series II
- iPDS (Personal Development System)
- minimum of 1 diskette drive

Required Software

ISIS Operating System or CP/M Operating System.

Documentation Package

One copy of each manual is supplied with the software package.

Description

BASIC-80 Reference Manual
BASIC Reference Book



MICROSOFT, INC. BASIC-80 INTERPRETER



Shipping Media

(Specify by Alpha Character when ordering.)

A—Single Density (Intellec compatible, IBM 3740/1 format)

B—Double Density (Intellec compatible, M²FM format)

F—Double-sided, Double Density 5¼" Floppy (iPDS format)

ORDERING INFORMATION

Order Code	Description
SD102CPM80A	Microsoft BASIC-80 Interpreter Software Package, CP/M version (Single-Density Diskette)
SD102CPM80B	Microsoft BASIC-80 Interpreter Software Package, CP/M version (Double-Density Diskette)
SD102CPM80F	Microsoft BASIC-80 Interpreter Software Package, CP/M version (Double-Sided, Double Density 5¼" Floppy) iPDS format
SD102ISS80F	Microsoft BASIC-80 Interpreter Software Package, ISIS version (Double-Sided, Double Density 5¼" Floppy) iPDS format

SUPPORT

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required.

*Microsoft is a trademark of Microsoft, Inc.

*CP/M is a registered trademark of Digital Research, Inc.

*MP/M-II is a trademark of Digital Research, Inc.



MICROSOFT INC.
BASIC-80 INTERPRETER



MICROSOFT*, INC. COBOL-80 SOFTWARE PACKAGE

- **GSA validation at low-intermediate level implementation of ANSI 74 requirements**
- **Combines the most useful Level 1 and Level 2 ANSI 74 features, making it the most extensive implementation of COBOL for microcomputers**
- **Interactive accept/display features allow direct interaction between data, program and operator**
- **Highly efficient pseudo-code compiler produces compact, fast code**
- **Complete utility software package, including macro assembler, linking loader, and MICROSOFT M/SORT**
- **Tailorable screen-formatting facilities allow configuration to any intelligent terminal for menu and forms generation**
- **COBOL offers four kinds of files for simplicity and speed in data retrieval**

COBOL has been the language of choice for commercial data processing for over two decades, and because of its established superiority, more useful business software has been written in COBOL than in any other language. That means quality, full-function packages, many written by the field's foremost experts, are available now. Existing systems for payroll, accounts receivable, general ledger, inventory, order entry, and forecasting can run under Microsoft COBOL-80.

COBOL offers the flexibility that makes products adaptable, versatile, and available to thousands of microcomputer users. COBOL-80 not only retains the high-level features of standard COBOL, but also introduces superior interactive capabilities and user-oriented features which represent a major advance in the commercial use of COBOL. With COBOL-80, direct interaction between data, program, and operator becomes possible. The immediate acknowledgement or rejection of user-entered data facilitates equally immediate corrections and modifications.

FEATURES

COBOL-80 ANSI Standard and GSA Validated

Because COBOL is so widely used, assurances of compatibility and portability are important to all users. The American National Standards Institute has established guidelines which provide a basis for making informed comparisons of different COBOL compilers. Elements of the COBOL language are allocated to twelve different functional processing modules, at two different levels. Microsoft COBOL combines the most useful Level 1 and Level 2 features.

The United States government, through the General Services Administration (GSA), tests COBOL compilers to validate their compatibility and their implementation of the ANSI 74 standard. Microsoft COBOL for the CP/M operating system has been

validated by GSA as a low-intermediate implementation of the ANSI 74 standard. Microsoft COBOL is approved for federal government installations, plus you have assurance that Microsoft COBOL will perform to specifications.

COBOL-80 supports such special features as:

Advanced verbs:

STRING, UNSTRING, COMPUTE, SEARCH,
PERFORM (VARYING/UNTIL)

Abbreviated and compound conditions
Sequential, Relative, and Indexed files
ASCII, packed and binary data formats
Runtime assignment of file names

Full COPY facility

Line Sequential files

Trace style debugging

COMP-3 data format

Program CHAIN

Program Segmentation

Formatted screen ACCEPT/DISPLAY

with a single command

Program Structuring

COBOL's unique suitability to the business environment is due in large part to the structuring capabilities built into the language. COBOL-80 programs have a natural, logical organization which reflects the nature of commercial data. This efficient, clean, top-down design makes COBOL-80 programs faster to write and easier to maintain than those written in other computer languages.

```
0000-MAIN-LINE.
    PERFORM 1000-INITIALIZE.
    PERFORM 2000-ENTER ORDER
    UNTIL END-SESSION.
    IF REPORT-REQUIRED
        PERFORM 3000-PRINT SUMMARY.
    PERFORM 4000-TERMINATE.
    CHAIN "MAINMENU".
```

```
1000-INITIALIZE.
    DISPLAY SIGN-ON-SCREEN.
    ACCEPT SIGN-ON-SCREEN.
    PERFORM 1100-CHECK-SECURITY-CODE.
    MOVE ZERO TO ORDER-COUNT
    ERROR-CODE.
    SESSION-TOTAL.
```

Data Structuring

The data in COBOL programs is arranged hierarchically. Information is stored in a logical structure with direct interconnection between related pieces of data. (Note here how COBOL's PICTURE specification allows exact decimal arithmetic for precise representation of any dollar amount.)

```
05 TRANSACTION
    10 TRAN-REF.          PIC X(12).
    10 TRAN-DATE.
        15 TRAN-MM       PIC XX.
        15 TRAN-DD       PIC XX.
        15 TRAN-YY       PIC XX.
    10 POST-DATE.
        15 POST-MM       PIC XX.
        15 POST-DD       PIC XX.
        15 POST-YY       PIC XX.
    10 TRAN-AMOUNT       PIC S9(8)V99.
    10 TRAN-AMOUNT       PIC X(4).
```

Once this structure has been coded it can be stored in a file and used in any program. Only one statement is needed to retrieve it: COPY TRANSDEF.COB.

COBOL-80 obviates the need to recode every program in a system when a single definition is added or

changed. Any chance of inconsistencies between programs is eliminated. At execution, each item in a COBOL-80 data structure may be referenced individually, or grouped items may be manipulated as easily. Processing code accesses the structure at any appropriate level:

```
MOVE JOURNAL-TRAN TO TRANSACTION.
MOVE CURRENT-DATE TO POST-DATE.
IF TRANS-AMOUNT < 0
    MOVE TRANSACTION TO CR-TRAN.
ELSE
    MOVE TRANSACTION TO DB-TRAN.
```

Screen Handling

COBOL's move from the batch environment to online applications has brought a new emphasis to the interaction between the application and the terminal operator. COBOL-80 provides a SCREEN SECTION for the definition of formatted screens. Special syntax is available for cursor positioning, protected and unprotected fields, highlighting, full and partial screen erase, and for defining connections between fields defined on the screen and data source/destination fields in WORKING-STORAGE. Data entry forms, menus, reports and other screens are ACCEPTed or DISPLAYed with a single command, so coding is simple. At execution time, related data items can be keyed, viewed together, and corrected before being entered into the program.

COBOL-80's screen handling facilities are Data General compatible, and learning to use the features is easy. Data descriptions are generally of the same form as in other sections of the Data Division. The screen section allows full screen descriptions without forcing allocation of WORKING-STORAGE space for unused portions of the screen.

On output, COBOL's extensive formatting capabilities provide neat, precise reports with minimal effort.

Program Chaining

COBOL-80's CHAIN verb facilitates interactive systems. Any number of separately coded applications or application segments can be reached through a main menu. These menu-driven applications which make ideal use of COBOL-80's interactive capabilities, allow smooth transfer of control from one program to another. With CHAIN, control is transferred from the menu program to any executable module as specified at runtime. COBOL-80 programs can also CALL COBOL-80 subprograms or Microsoft FORTRAN-80 or assembly language subroutines.

File Handling

COBOL-80 aims for simplicity and speed in data retrieval. COBOL-80 offers four kinds of files, so data storage can aptly correspond to the application for which it will be used. Each file type has unique characteristics:

Sequential

The fastest possible access to test, packed, and binary data in any combination.

Line Sequential

Text data in line format. Compatible with the files generated by many text editors.

Relative

Random access by record number. Direct disk access makes relative files extremely fast, yet data can be accessed in any sequence, deleted or updated interactively, and cross-referenced by key across files.

Microsoft COBOL-80 and the ANSI Standard

Module

Nucleus

Features of Microsoft COBOL

All of Level 1, plus these features of Level 2:

CONDITIONS:

Level 88 conditions with value series or range
Use of logical AND/OR/NOT in conditions
Use of algebraic

relational symbols (<, >, =)

Implied subject, or both subject and relation, in relational conditions

Sign Test

Nested IF statements; parentheses in conditions

Indexed Sequential

The most powerful data access method available from any data processing language in any environment. A field within each record is chosen as a key, and the value of this key identifies a record to be read, written, updated, or deleted.

Segmentation

COBOL-80's program segmentation capability makes maximum use of memory when large programs are executed. Segmentation brings individual program sections into memory as they are needed. This allows execution of programs whose size may exceed machine memory by several times. Program segmentation is easily implemented by assigning a single number to each program section which indicates whether the section is to be resident in memory or overlaid during execution. Any section to be overlaid is automatically read into a preallocated section of memory during execution.

VERBS:

Extensions to the functions of ACCEPT and DISPLAY for formatted screen handling

ACCEPTance of data from DATE/DAY/TIME STRING and UNSTRING statements
COMPUTE with multiple receiving fields
PERFORM VARYING

IDENTIFIERS:

Mnemonic-names for ACCEPT or DISPLAY devices
Procedure-names consisting of digits only
Qualification of names (in Procedure Division statements only)

Sequential, Relative, and Indexed I/O	All of Level 1 plus these features of Level 2: RESERVE clause Multiple operands in OPEN & CLOSE, with individual options per file Value of FILE-ID is data- name	Library Inter-Program Communication Table Handling Debugging	Level 1 Level 1 All of Level 1 Full Level 2 formats for SEARCH statement Special extensions to ANSI-74 Standard providing convenient trace style debugging. Conditional Compilation: lines with "D in column 7" are bypassed unless "WITH DEBUGGING MODE" is given in SOURCE-COMPUTER paragraph.
Sequential I/O	EXTEND mode for OPEN WRITE ADVANCING data name lines LINAGE phrase and AT END-OF-PAGE clause		
Relative and Indexed I/O	DYNAMIC access mode (with READ NEXT) START (with key relationals EQUAL, GREATER, or NOT LESS)	Segmentation	Level 1

Utility Software Package

COBOL-80 includes the Microsoft Utility Software Package. The Utility Software Package includes the MACRO-80 macro assembler, the LINK-80 linking loader, and the CREF-80 Cross-Reference Facility. Refer to the description of the Utility Software Package for full details.

M/SORT

The COBOL-hosted version of M/SORT (M/SORT-C) is a record-sorting facility available to the programmer through the 1974 ANSI COBOL SORT/MERGE

statements. For M/SORT-C, the source of the input records may be one or a set of disk files; or, records may be constructed in memory by a user-written COBOL procedure and RELEASED to M/SORT one at a time.

Similarly, the sorted output records may be automatically written to a disk file; or, records may be left in memory for processing by a user-written OUTPUT PROCEDURE within the COBOL program. M/SORT-C can load an indexed sequential file.

M/SORT-C provides for a special SORT STATUS register which can collect and return any errors encountered during a sort.

SPECIFICATIONS

Operating Environment

COBOL-80 requires about 40K bytes of user memory in addition to the operating system's space. COBOL object programs will run on systems as small as 32K bytes.

Required Hardware

Intellec Microcomputer Development System
—Model 800 or
—Series II
—iPDS (Personal Development System)
—minimum of 1 diskette drive

Required Software

CP/M* Operating System or MP/M-II* Operating System.

Documentation Package

One copy of each manual is provided with the software package.

Description
COBOL-80 Reference Manual
COBOL-80 User's Guide
Microsoft Utility Software Manual
M/SORT Reference Manual



Shipping Media

(Specify by Alpha Character when ordering.)

A—Single density (Intellec compatible, IBM 3740/1 format)

B—Double density (Intellec compatible, M²FM format)

F—Double-sided, Double Density, 5 1/4 " diskette (iPDS format)

ORDERING INFORMATION

Order Code	Description
SD104CPM80A	Microsoft COBOL-80 Software Package, CP/M version (Single-Density Diskette)
SD104CPM80B	Microsoft COBOL-80 Software Package, CP/M version (Double-Density Diskette)
SD104CPM80F	Microsoft COBOL-80 Software Package, CP/M version (iPDS Format)

SUPPORT:

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

Required Software	Operating Environment	Required Hardware
CP/M Operating System or MP/M-II Operating System	COBOL-80 requires about 40K bytes of user memory in addition to the operating system's space. COBOL object programs will run on systems as small as 32K bytes.	Intel 8080 or 8085 Microcomputer Development System —Model 800 or —Series II —iPDS (Personal Development System) —minimum of 1 diskette drive
Documentation Package		
One copy of each manual is provided with the software package.		

An Intel Software License required.
*Microsoft is a trademark of Microsoft, Inc.
*CP/M is a registered trademark of Digital Research, Inc.
*MP/M-II is a trademark of Digital Research, Inc.



MICROSOFT INC.
FORTAN-80



MICROSOFT*, INC. FORTRAN-80 SOFTWARE PACKAGE

- **Full ANSI standard FORTRAN X3.9-1966 (except the complex data type)**
- **Provides numerous enhancements to the 1966 ANSI standard**
- **Compiles several hundred statements per minute in a single pass**
- **Optimizes the generated object code by common subexpression elimination, peephole optimization, constant folding, and branch optimizations**
- **Provides diagnostic output: descriptive error messages, error summaries, and fully symbolic listings of generated machine language**
- **Supplies an extensive library of efficient subroutines**
- **Supports user-written non-standard I/O drivers for each logical unit number to interface non-standard devices to FORTRAN programs**

Microsoft FORTRAN-80 brings the most popular science and engineering programming language to 8080/8085 microcomputers. FORTRAN-80 is comparable to FORTRAN compilers on large mainframes and minicomputers. The FORTRAN-80 package includes full ANSI Standard FORTRAN X3.9-1966 except the COMPLEX data type. With this compiler, users may take advantage of many applications programs already written in FORTRAN.

FEATURES

FORTAN-80 enhances the 1966 ANSI Standard in several ways:

- Single-byte LOGICAL variables which can be used as integer quantities in the range +127 to -127.
- DO loops which use LOGICAL variables for tighter, faster execution of small loops.
- Mixed-mode arithmetic expressions.
- Hexadecimal constants.
- Hollerith (character) literals accepted.
- Logical operations on integer data. .AND., .OR., .NOT., .XOR., can be used for 8-bit, 16-bit, or 32-bit Boolean operations.
- READ/WRITE End-of-File or Error-Condition transfer. END=n and ERR=n (where n is the statement number) can be included in READ or WRITE statements to transfer control to the statement specified by n when an error or end-of-file is detected.
- ENCODE/DECODE for FORMAT operations to memory.
- IMPLICIT statement for redefining default variable types by specifying a type and a range of initial letters.
- INCLUDE statement for including commonly used subroutines, code, or declarations from another file.
- INTEGER*4 variables and constants using 32 bits in the range of +2,147,483,647 to -2,147,483,648.
- Support for CP/M* version 2.x providing access to a maximum of 65,535 random records in a file as large as 8 megabytes.

COMPILER DESCRIPTION

FORTRAN-80 compiles several hundred statements per minute in a single pass. It requires no more than 27K bytes of memory to compile most programs. Additional memory, when available, is used for symbol table storage and optimizations. Compiled programs are relocatable modules that are linked and loaded at runtime.

The FORTRAN-80 compiler optimizes generated object code in several ways:

Common Subexpression Elimination

Common subexpressions are evaluated once; the value is automatically substituted in subsequent occurrences of the subexpression.

Peephole Optimization

In special cases, small sections of code can be replaced by more compact code.

Example: $I=I+1$ uses an INX H instruction instead of a DAD.

Constant Folding

Integer constant expressions are evaluated at compile time.

Branch Optimizations

The number of conditional jumps in arithmetic and logical IFs is minimized.

The compiler also provides diagnostic output. Descriptive error messages include the preceding twenty characters. At program's end, the compiler generates an error summary. A fully symbolic listing of the generated machine language is also produced. This is supplemented by tables of addresses assigned to labels, variables, and constants.

SPECIFICATIONS

Operating Environment

The FORTRAN-80 compiler occupies approximately 24K bytes of memory, using the CP/M operating system. Most programs can be compiled within 27K bytes of additional memory. At link time, the LINK-80 linking loader occupies about 14K bytes. The Runtime Library requires 2K-14K bytes, depending on the type of program being run.

Subroutine Library

FORTRAN-80 supplies an extensive library of efficient subroutines. Only the necessary subroutines are loaded by the linker at load time. The FORTRAN library includes the following Intrinsic Functions:

ABS	DATAN	DSIN	MAX1
AINT	DATAN2	DSORT	MIN0
ALOG	DBLE	EXP	MIN1
ALOG10	DCOS	FIX	MOD
AMAX0	DEXP	FLOAT	OUT
AMAX1	DIM	IABS	PEEK
AMIN0	DLOG	IDIM	POKE
AMIN1	DLOG10	IDINT	SIGN
AMOD	DMAX1	INP	SIN
ATAN	DMIN1	INT	SNGL
ATAN2	DMOD	ISIGN	SQRT
COS	DSIGN	MAX0	TANH
DABS			

The library also contains efficient routines for 16-bit and 32-bit integer arithmetic and 32-bit and 64-bit floating point arithmetic.

Utility Software Package

The FORTRAN-80 package includes the Microsoft Utility Software Package. The Utility Software Package includes the MACRO-80 macro assembler, the LINK-80 linking loader, and the CREF-80 Cross-Reference Facility. Refer to the description of the Microsoft Utility Software Package for full details.

Custom I/O Drivers

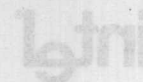
Users may write non-standard I/O drivers for each Logical Unit Number, so interfacing non-standard devices to FORTRAN programs is straightforward.

Required Hardware

Intel Microcomputer Development System
—Model 800 or
—Series II
—iPDS (Personal Development System)
—minimum of 1 diskette drive

Required Software

CP/M* Operating System or MP/M-II* Operating System.



Documentation Package

One copy of each manual is provided with the software package.

Description

FORTRAN-80 Reference Manual
FORTRAN-80 User's Manual
Microsoft Utility Software Manual

Shipping Media

(Specify by Alpha Character when ordering.)

- A—Single density (Intellec compatible, IBM 3740/1 format)
- B—Double density (Intellec compatible, M²FM format)
- F—Double-sided, Double Density 5 1/4" Floppy (iPDS format)

ORDERING INFORMATION

Order Code	Description
SD103CPM80A	Microsoft FORTRAN-80 Software Package, CP/M version (Single-Density Diskette)
SD103CPM80B	Microsoft FORTRAN-80 Software Package, CP/M version (Double-Density Diskette)
SD103CPM80F	Microsoft FORTRAN-80 Software Package, CP/M version (iPDS Format)

SUPPORT

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required.
*Microsoft is a trademark of Microsoft, Inc.
*CP/M is a registered trademark of Digital Research, Inc.
*MP/M-II is a trademark of Digital Research, Inc.



MICROSOFT INC.
FORT WORTH



MICROSOFT*, INC. BASIC-86 INTERPRETER SOFTWARE PACKAGE

- Compatible with other Microsoft BASIC compilers and interpreters
- Sophisticated string handling and structured programming features for applications development
- Direct transfer of BASIC programs to the 8080, 8085 and 8088
- Random and sequential file manipulation where random file record length is user-definable
- Read or write memory location capabilities
- Meets the requirements for the ANSI subset standard for BASIC, and supports many enhancements
- Extensive text editing features built-in
- Automatic line number generation and renumbering
- Supports assembly language subroutine calls
- Trace facilities for easier debugging

BASIC Release 5.0 from Microsoft is an extensive implementation of BASIC. Microsoft BASIC gives users what they want from a BASIC—ease of use plus the features that are comparable to a minicomputer or large mainframe.

BASIC-86 meets the requirements for the ANSI subset standard for BASIC, as set forth in document BSRX3.60-1978. It supports many unique features rarely found in other BASICs.

FEATURES

- Four variable types: Integer (–32768, +32767), String (up to 255 characters), Single-Precision Floating Point (7 digits), Double-Precision Floating Point (16 digits).
- Trace facilities (TRON/TROFF) for easier debugging.
- Error trapping using the ON ERROR GOTO statement.
- PEEK and POKE statements to read or write any memory location.
- Automatic line number generation and renumbering, including referenced line numbers.
- Matrices with up to 255 dimensions.
- Boolean operators OR, AND, NOT, XOR, EQV, IMP.
- Formatted output using the PRINT USING facility, including asterisk fill, floating dollar sign, scientific notation, trailing sign, and comma insertion.
- Direct access to I/O ports with the INP and OUT functions.
- Extensive program editing facilities via EDIT command and EDIT mode subcommands.
- Assembly language subroutine calls (up to 10 per program) are supported.
- IF/THEN/ELSE and nested IF/THEN/ELSE constructs.
- Supports variable-length random and sequential disk files with a complete set of file manipulation statements: OPEN, CLOSE, GET, PUT, KILL, NAME, MERGE.

BASIC-86 Commands, Statements, Functions

AUTO	RENUM	NAME
LIST	WIDTH	SAVE
NULL	CONT	EDIT
TROFF	MERGE	NEW
CLEAR	RUN	TRON
LOAD	DELETE	

Program Statements

CALL	RANDOMIZE	RETURN
GOSUB	COMMON	WAIT
END	DEF FN	ON GOSUB
GOTO	ERROR	DIM
STOP	POKE	FOR/NEXT/
WHILE/	RESUME	STEP
WEND	SWAP	IF/THEN/
CHAIN	DEFDBL	ELSE
DEF USR	DEFSTR	ON ERROR
LET	DEFSGN	GOTO
REM	DEFINT	OPTION BASE

Input/Output Statements and Functions

CLOSE	GET	NAME
KILL	POS	PUT
OUT	FIELD	EOF
RESTORE	LSET/RSET	SPC
READ	PRINT	INKEY\$
TAB	USING	INPUT
DATA	LOC	OPEN
LINE	MK1\$	CVD
INPUT	MKS\$	CVI
PRINT	MKD\$	CVS
WRITE	LLIST	
LPRINT	LPOS	

Arithmetic Functions

ABS	SIN	LOG
INT	CDBL	FIX
SGN	CSNG	COS
ATN	CINT	RND
EXP	SQR	TAN

String Functions

ASC	STR\$	INSTR
LEN	HEX\$	RIGHT\$
STRING\$	OCT\$	MID\$
CHR\$	VAL	SPACES
LEFT\$		

Operators

=	*	XOR
^	<=	NOT
<	=	EQV
>	<>	MOD
-	\	IMP
/	=	OR
		AND

Special Functions

ERL	ERR	VARPTR
USR	FRE	PEEK

SPECIFICATIONS

Operating Environment

The standard disk version of Microsoft BASIC-86 occupies 32K bytes of memory. Microsoft BASIC-86 Interpreter is compatible with the CP/M-86* operating system.

Required Hardware

Microsoft BASIC-86 Interpreter will operate with any currently supported configuration of CP/M-86 or MP/M-86*.

Required Software

CP/M-86* Operating System or MP/M-86* Operating System.

Documentation Package

One copy of each manual is provided with the software package.

Description

BASIC-86 Reference Manual
BASIC Reference Book

Shipping Media

(Please specify by Alpha Character when ordering.)

A—Single Density (Intellec compatible, IBM 3740/1 format)

ORDERING INFORMATION

Order Code	Description
SD203CPM86A	Microsoft BASIC-86 Interpreter Software Package, CP/M-86 version (Single-Density Diskette)
SD203RMX86D	Microsoft BASIC-86 Interpreter Software Package, iRMX/86 version (Single-Density Diskette)

SUPPORT

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required.

*Microsoft is a trademark of Microsoft, Inc.

*CP/M-86 is a registered trademark of Digital Research, Inc.

*MP/M-86 is a trademark of Digital Research, Inc.

JOVIAL-86 CROSS-COMPILER

- DEC*-10 host (TOPS-10 O/S)
- IBM 370, 3033, 4341 host (OS/MVS)
- Full MIL-STD-1589B language implementation
- Extensively optimized object code
- Supports IEEE floating-point math with 8087 coprocessor
- RAM/ROM separation of constants and code supported
- Object compatible and linkable with Intel's languages for 8086/8088
- Generates source symbols, lines and type information for debugging with ICE™-86A and PSCOPE™
- Type-checking of parameters between modules optionally enforced by the linker
- Choice of silicon or software implementation for floating point

JOVIAL was developed in the late 1950s by Jules Schwartz and colleagues to aid in programming large complex realtime systems. The language was adopted by the U.S. Air Force in 1967 and its use is now required for programming of all avionics embedded computer systems.

Intel's JOVIAL Compiler implements the latest published standard of this language (MIL-STD-1589B) for iAPX 86/20 16-bit microprocessors. Intel's implementation of J73 offers a complete solution for programming applications in JOVIAL from compiling modules on the mainframe to downloading software to Intel's Series III development system where a host of JOVIAL-86-compatible languages, relocation and linkage software, symbolic debuggers and an in-circuit emulator (ICE-86A) are available.

FEATURES

The JOVIAL language was designed for programming large complex realtime systems. The language is a derivative of IAL (International Algebraic Language) and has borrowed compound statement and control structures from it. JOVIAL supports fixed and floating-point, signed and unsigned integers, strings, tables, status items and bits. The language provides access to almost any configuration of logical or arithmetic values including packed structures of flexible description. The communication pool (COMPOOL) permits sharing of system data among many subprograms by providing a centralized data and procedure description which enforces uniform usage and simplifies centralized configuration management of a JOVIAL system data base.

JOVIAL-86 COMPILER DESCRIPTION

The JOVIAL-86 compiler operates in multiple passes. The front-end phase analyzes the input program and

translates it into an intermediate language. The global optimizer phase analyzes the intermediate code and applies extensive optimizations such as constant and value folding, multiply distribution and operator reassociation for efficient address calculation, compile time short-cut booleans, unreachable code deletion, and unnecessary store suppression. The code generator phase generates relocatable object code from this intermediate code. Other functions in the final phase are responsible for generating listings and COMPOOL output.

The compiler supports development of complex and large applications by providing a comprehensive cross-reference listing that identifies the statement number where the variable is set. JOVIAL-86 can also generate a statistical listing showing the distribution of various symbol table classes and intermediate language operators. The assembly language output listing displays the object program in assembly mnemonics. Optionally, the compiler can produce a reformatted source program.

PROGRAM DEVELOPMENT PROCESS

The JOVIAL source programs may be developed and maintained on the mainframe IBM 370 or DEC-10 (and compatible computers), or alternately, the source programs may be created and maintained on the Intellec Series III Microcomputer Development System, sent to the mainframe for compilation and brought back to the Series III for further processing. This processing includes relocating, linking, debugging, emulating and executing. The ONLINE utility allows the Series III to act as a terminal to the mainframe to edit and examine files, etc. The transfer of source and object files between the mainframe and the Series III is accomplished with either the 2780/3780 emulator or the upload/download software on the Series III.

The object code produced by the compiler is in Intel's standard object module format and is, therefore, compatible with standard Series III utilities like LINK86, LOC86, LIB86 and OH86. Adequate compatibility with subprograms written in PASCAL86, FORTRAN86, ASM86 and PL/M-86 is assured by following consistent and well-documented parameter passing and register usage conventions. The compiler generates source program symbols, lines and type information in the object code to allow symbolic debugging with the Series III debugger and in-circuit emulator.

The compiler generates 8087 instructions for floating-point operations. In the absence of the 8087 processor, the 8087 emulator software may be linked with the JOVIAL program to execute the floating-point instructions.

SPECIFICATIONS

Operating Environment

Intel's JOVIAL-86 compiler runs under MVS on IBM 370 and requires a minimum 750 Kb partition. The DEC-10 version runs under TOPS-10 operating system and requires 100K words.

Required Hardware

- DEC-10, IBM 370 or compatible computer
- Intellec Series III Microcomputer Development System
- Necessary Modems and Cables for Serial Link

Required Software

- Series III Operating System
- iAPX86 Family Utilities

Optional Software

- Mainframe Link (2780/3780 Emulator)

Documentation Package

One copy of each manual is provided with the software package. Additional copies may be ordered.

Part No.	Description
121886	JOVIAL Language Specification
121887	JOVIAL-86 User's Guide
172174	Asynchronous Communications Link User's Guide (DEC-10 only)

Shipping Media

- One phase-encoded, 1600 B.P.I. Magnetic Tape
- JOVIAL-86 Compiler Object Code (DEC-10 and IBM)
- Asynchronous Communications Link Software (DEC-10 only)
- One Single- and one Double-Density Diskette (ISIS II)
- JOVIAL-86 Run Time Library (DEC-10 and IBM)
- Asynchronous Communications Link Software (DEC-10 only)
- On-line Utilities (DEC-10 and IBM)

ORDERING INFORMATION

Order Code	Description
SD900TOP86	JOVIAL-86 Cross-Compiler Package for DEC ^{SYSTEM-10} *
SD900MVS86	JOVIAL-86 Cross-Compiler Package for IBM 370

Requires Software License

*DEC and DEC^{SYSTEM-10} are registered trademarks of Digital Equipment Corporation.

SUPPORT

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, and Monthly Technical Newsletters are available.